BBN Systems and Technologies Corporation

A Subsidiary of Bolt Beranek and Newman Inc.

AD-A201 910

Report No. 6937

Research in Knowledge Representation For Natural Language Communication and Planning Assistance

Final Report

18 March 1985 to 30 September 1988

B. Goodman, B. Grosz, A. Haas, D. Litman, T. Reinhardt, C. Sidner, M. Vilain, R. Weischedel, C. Whipple

Prepared for:

Defense Advanced Research Projects Agency

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 6937	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Research in Knowledge Representation for Natural Language Communication and Planning Assistance - Final Report		5. TYPE OF REPORT & PERIOD COVERED Final Report 3/85 - 9/88 6. PERFORMING ORG. REPORT NUMBER 6937
7. AUTHOR(s) B. Goodman, B. Grosz, A. Haas, D. Litman, T. Reinhardt, C. Sidner, M. Vilain, R. Weischedel, C. Whipple		8. CONTRACT OR GRANT NUMBER(s) N00014-85-C-0079
9. PERFORMING ORGANIZATION NAME AND ADDRESS BBN Systems and Technologies Corporation 10 Moulton Street Cambridge, Ma. 02138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Department of Navy Arlington, Va. 22217		12. REPORT DATE November 1988 13. NUMBER OF PAGES 194
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial Intelligence, Discourse, Domain Model, Intelligent Interfaces, Knowledge Representation, Natural Language Understanding, Parallel Processing, Planning, Plan Recognition, Propositional Attitudes, Reasoning, Semantics, Speech Understanding, Truth Maintenance System

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

BBN's DARPA project in Knowledge Representation for Natural Language Communication and Planning Assistance has two primary objectives: (1) To perform research on aspects of the interaction between users who are making complex descisions and systems that are assisting them with their task. In particular, this research is focused on communication and the reasoning required for performing its underlying task of discourse processing, planning and plan recognition and communication repair. (2) Based

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

on the research objectives to build tools for communication, plan recognition, and planning assistance and for the representation of knowledge and reasoning that underlie all of these processes.

This final report summarizes BBN's research activities performed under this contract in the areas of knowledge representation and speech and natural language. In particular, the report discusses our work in the areas of knowledge representation, planning, and discourse modelling. We describe a parallel truth maintenance system. We provide an extension to the sentential theory of propositional attitudes by adding a sentential semantics. The report also contains a description of our research in discourse modelling in the areas of planning and plan recognition. We describe a new compositional form of plan recognition that integrates planning and plan recognition. We also discuss multi-agent collaboration in the planning process to develop a shared plan. The report also describes recent attempts at generalizing the modelling of domains that is central to speech and natural language processing. We describe as part of that effort an experiment using a dictionary to build a taxonomy of primitive concepts. We conclude by documenting publications and presentations by members of the research group over the course of the contract.

Keywords: artificial intelligence, intelligent interface, parallel processing. (KR)

RESEARCH IN KNOWLEDGE REPRESENTATION FOR NATURAL LANGUAGE COMMUNICATION AND PLANNING ASSISTANCE

Final Report
18 March 1985 to 30 September 1988

Principal Investigator:
Dr. Bradley A. Goodman

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

ARPA Order No. 3414

Contract No. N00014-85-C-0079

Effective Date of Contract
18 March 1985

Contract Expiration Date
30 September 1988

Amount of Contract
$3,476,702

Scientific Officer
Dr. Alan R. Meyowitz

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-85-C-0079. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Research Projects Agency or the U.S. Government.

TABLE OF CONTENTS

1. Executive Summary

This chapter summarizes the research directions followed during the course of the contract and highlights some of our accomplishments. The Knowledge Representation for Natural Language Communication and Planning Assistance contract grew out of a long history of DARPA supported knowledge representation and natural language contracts at BBN beginning in 1977. Those contracts provided many advances in the areas of knowledge representation and natural language that set the stage for the research conducted during the course of this contract. The earlier research as well as that conducted under this contract is now being applied not just to natural language systems but also to spoken language systems.

1.1 Research Focus

In this section we lay out the primary areas of research that were investigated under this contract.

1.1.1 Theme

The research plan we followed in this contract was aimed toward fundamental problems of Knowledge Representation and Reasoning relevant to Spoken and Natural Language Communication and Planning Assistance. Central to this plan was our research in the representation of plans, plan recognition, plan formation, reasoning about plans and actions, and modelling the discourse. The exploration of these research topics required

investigating both short-term as well as long-term solutions. We transferred some of our research results to other DARPA-supported activity at BBN, such as the BBN spoken language system (SLS).

1.1.2 Spoken and Natural Language Communication

Central to extension of spoken or natural language understanding systems from handling single (isolated) utterances to coherent dialogue is the modeling of discourse. Plans that represent the intentions of the speaker and listener are a significant component of such a model. The representation of user beliefs is an important constituent of these plans. The process of inferring the user's plans from the user's utterances (plan recognition) is the key to understanding dialogues. To further our research in understanding the intentions of utterances we investigated the ability to understand in the face of miscommunication. Reference identification is another important element of spoken and natural language processing that benefits from discourse modelling. Reference provides a handle on the interpretation of utterances by determining the objects in the world being referred to by the noun phrases and pronouns.

1.1.3 Planning Assistance

Planning is another element underlying natural language communication. A major thrust of our research was the extension of knowledge representation systems for representing beliefs of agents, actions, time, continuous processes, partial hypothetical plans and multiple agents. The knowledge representation and inference mechanisms developed are useful not just for work in planning but they provide the fundamental underpinnings to

parts of our spoken language system. In conjunction with our knowledge representation research, we also explored the equally critical area of knowledge acquisition.

1.2 Significant Results during the Course of the Contract

Our research during the past three and one-half years has addressed major aspects of these problems resulting in some significant results.

We developed the KL-Two hybrid knowledge representation language, integrating the NIKL representation system with RUP ("Reasoning Utility Package") through the PENNI interface.

We developed a new representation formalism that includes fluids modelled in a discrete manner based on a notion called granules and processes that are continuous and modelled discretely in a manner that permits serial as well as concurrent composition.

We have developed a logic that permits representation of nested beliefs of several agents, allows quantification with various scoping, and has efficient reasoning based on first-order unification.

As an initial exercise in parallel programming, we have developed a parallel unification based parser for a grammar, a grammar for English with excellent coverage, and a running program on a Vax, Symbolics and a parallel version on the BBN Butterfly. A serial version of this parser became the basis of the CFG parser used in the BBN spoken language system.

We developed a richly expressive intensional logic language for capturing the semantics of natural language sentences, including modality, tense and context-dependence. This semantic is being employed in the BBN spoken language system.

We investigated autoepistemic reasoning, especially its relation to other kinds of non-monotonic reasoning.

We developed a model of discourse structure including attention an intention.

We extended our discourse theory to model collaborative interaction between a user and a system. Knowledge of shared plans is incorporated in the model.

We analyzed and understood miscommunication phenomena, surrounding the use of noun phrase references, in actual videotaped dialogues.

We demonstrated a reference mechanism based on relaxation matching and implemented in the KL-Two knowledge representation language.

We implemented an incremental "keyhole" plan recognizer. We then modified it from "keyhole" recognition to "intended" recognition so that it fits in with our model of incremental intended plan recognition.

We integrated our plan recognition algorithm into a data base and graphics interface to demonstrate how plan recognition facilitates communication and information retrieval.

We explored issues relating to the representations of plans, keeping in mind the dual use of plans for plan recognition and planning. This involved providing plan representation with a richer semantics and a representation for actions.

We began expanding our current plan recognition algorithm to handle miscommunication situations. That expansion involves developing a "contructive" plan recognition algorithm that ties together the contruction and recognition of plans, i.e., it can build the plans that it will recognize. This allows one to recognize novel plans as well as helping detect and recover from miscommunication of intention in the recognition process.

We began our investigation of critical dimensions of such a constructive model of plan recognition by looking at cognitive science models of planning and problem solving in complex and ill-structured domains.

We began the process of integrating our discourse components into a spoken-language system. In particular, we began the process of adding reference identification and plan recognition components to such a system.

We explored the acquisition of knowledge for domain modelling for speech and natural language systems.

1.3 Overview of this Report

We provide in the rest of the report articles that describe a cumulation of our research under the project. Chapter 2 describes a multiprocessor-based truth maintenance system that was developed for a MIMD (multiple instruction/multiple data) multiprocessor. We discuss the algorithms underlying the system and strategies for increasing the concurrency of the system. In Chapter 3, we provide an extension to the sentential theory of propositional attitudes by adding a sentential semantics. We use this extension to present a formal semantics for a fragment of English. Chapter 4 describes a compositional form of plan recognition, constructive plan recognition, that removes the completeness assumption imposed by many standard plan recognition algorithms and that is conducive to the detection and correction of miscommunication in user's plans. The new formulation addresses the issue of novel plans making it an ideal candidate for providing robustness in plan-based systems. In Chapter 5, we discuss intelligent agents that can collaborate with other agents in the design and performance of plans to achieve shared objectives. We describe how to extend multi-agent planning systems to allow for collaboration in the planning process. Chapters 6 and 7 describe some recent attempts at generalizing the modelling of domains that is central to speech and natural language processing. We illustrate one such attempt by describing an experiment we carried out using a dictionary to build a taxonomy of primitive concepts. Finally, in Chapter 8, we detail the publications and presentations made by our group during the course of the project.

2. A Parallel Truth Maintenance System

Marc Vilain, BBN Systems and Technologies Corporation

Abstract: This paper is concerned with multiprocessor-based truth maintenance systems, focussing on a monotonic single-context system. MIMD algorithms underlying this system are described, and their theoretical characteristics are discussed. Strategies for increasing the concurrency of the system are then considered in detail. Finally, these results are considered in the light of alternative models of truth maintenance and of alternative parallel architectures.

2.1 Introduction

The truth maintenance system, or TMS, is a common artificial intelligence programming technique. This widespread acceptance, however, is matched by widespread disagreement over optimal truth maintenance strategies. The TMS literature describes systems that vary along dimensions of monotonicity, complexity of support constraints, incrementality of context switches, and others. Disagreement also exists as to how and how well truth maintenance systems may be implemented on parallel hardware.

One factor contributing to this disagreement is the plurality of competing TMS models that might be offered for parallel implementations. The discussion is further muddled by the converse factor: the large number of multiprocessor architectures in which a parallel TMS might be embedded. In this paper, I do not attempt to resolve the issue in general, but

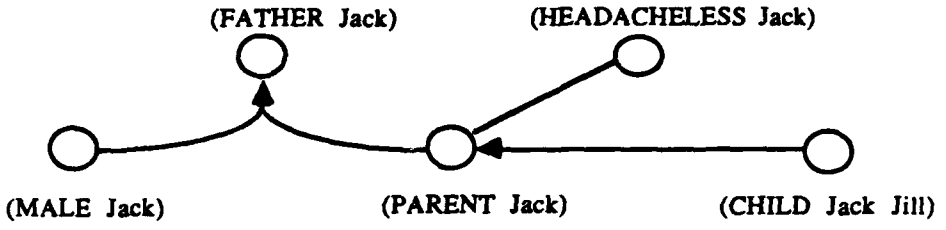
rather focus on a particular TMS , as implemented on a particular machine. The TMS in question is a single-context monotonic system derived from McAllester's RUP [McAllester 80, 84]. The hardware on which I've been exploring parallel versions of this system is the Butterfly™ multiprocessor, a MIMD (multiple instruction/multiple data) machine that supports up to 256 independent processors with fully shared memory.

The results of this study are briefly as follows. Although unfavorable worst case bounds exist on the degree to which a TMS may be made parallel, natural sources of concurrency arise in the TMS . These range from the straightforward (e.g., adding simultaneous assertions) to the unexpected (e.g., running the TMS concurrently with the program in which it is embedded). Additionally, truth maintenance in this TMS is equivalent to constraint propagation in RCLP [Mackworth 77], and these results generalize to this problem and to its numerous analogues. Finally, these results carry over to some degree to other TMS models, and to other MIMD architectures, but it is unclear how applicable they are to SIMD (single instruction/multiple data) models of parallel computing.

2.2 The problem and its properties

Abstractly, the role of a TMS is to maintain consistent a program's interdependent "beliefs". In McAllester's model, which my work takes as a premise, beliefs are simply ground propositions (predicate calculus expressions with no quantifiers or variables). The interdependencies between beliefs are expressed as a database of logical clauses, each of which is a disjunct of literals, i.e.,positive or negative propositions.

Figure 1: Propositional Constraint Graph



Clauses:

(not (CHILD Jack Jill)) or (PARENT Jack)
(not (PARENT Jack)) or (not (HEADACHELESS Jack))
(not (MALE Jack)) or (not (PARENT Jack)) or (FATHER Jack)

i. e.,(CHILD Jack Jill) -> (PARENT Jack)
(not ((PARENT Jack) and (HEADACHELESS Jack)))
((MALE Jack) and (PARENT Jack)) -> (FATHER Jack)

N.B.: Arrowheads of links in graph indicate positive literals in constraints.

Propositions have a truth status which is drawn from the conventional set *true* and *false*, augmented by the undetermined truth value *unknown*[1]. Truths are either assigned to propositions by user assertions, or as a result of truth maintenance, which in this TMS is simply deduction. Deduction is readily implemented here by Waltz-style constraint propagation [Waltz 75]. To each proposition corresponds a node in a constraint graph, and to each clause corresponds a constraint link; see Figure 1 for examples. Constraints are propagated when the truths of the propositions in all but one literal in a clause become known: if the literals are all unsatisfied,[2] the clause is then used to deduce the truth of the remaining literal.

[1] This differs from such non-monotonic formalisms as Doyle's, in which the belief status of a proposition is either *in* or *out*, marking the proposition respectively as having, or lacking, reason to be believed [Doyle 78].

[2] A literal is unsatisfied if it requires of a proposition a truth value opposite of the one it has been assigned by assertion or as the result of deduction.

McAllester [McAllester 80] notes that constraint propagation is complete for his formulation of truth maintenance when the database of constraints meets certain restrictions. In particular, if all the clauses in the database are HORN (all the literals but one are negative) or NOGOOD (all the literals are negative), constraint propagation is a sound and complete polynomial time decision procedure for the TMS. In the general case, however, constraint propagation is incomplete. If arbitrary clauses are allowed in the TMS, the problem is NP-complete, by a straightforward reduction from SAT.

McAllester additionally notes that his TMS is essentially performing unit clause resolution. This is a variant of propositional resolution in which any clause of size $n \geq 2$ can only be resolved against *n-1* "unit" clauses of size *1*. This is of special interest in considering parallel implementations of this TMS, since it leads to an unfavorable worst case result. Indeed, unit clause resolution is P complete [Jones & Laaser 77][3]. Problems in this class are strongly conjectured not to have a parallel solution that is guaranteed to terminate in less than polynomial time (see [Reif 85]). Hence, the asymptotic worst-case performance of any parallel algorithm for McAllester's TMS can not be expected to improve on that of the best serial solution. In fact, Kasif [Kasif 86] shows that a generalization of TMS constraint propagation, the relaxed consistent labelling problem [Mackworth 77], is P complete as well.

Worst-case results such as these do not render vacuous the enterprise of parallel truth maintenance. Rather, they bring to light the need for parallel TMS strategies that exploit sources of concurrency that are independent of any inherent TMS sequentiality. This is the

[3]If a problem π is P complete, then any problem in P is reducible to π by a transformation requiring logarithmic working space. In [Garey & Johnson 79], this class of problems is also referred to as *log-space complete for P*.

principal focus of this work, and I will return to it after first considering the nature of parallelism on the Butterfly multiprocessor.

2.3 The Butterfly and Butterfly LISP

The Butterfly, manufactured by BBN, is a MIMD machine configurable with up to 256 processors. Each of the processors may run an independent program (the MI part of MIMD -- multiple instruction streams) in its own local memory (the MD part -- multiple data). Particular to this machine is that each processor's memory is made entirely accessible to all other processors through a network of butterfly switches. This architecture has several implications to the programmer. First, very general forms of concurrency may be developed, but not without the scheduling cost of assigning processors to tasks. Second, since the butterfly switch imposes a memory access delay, allocation strategies for global variables must compromise between a processor's rapid access to local variables and delayed access to variables reached through the switch. Finally, since the processors are independent, the programmer must contend with simultaneous attempts to set variables, and the many locking considerations that ensue.

Butterfly LISP [Steinberg *et al.* 86] addresses these programming issues with a number of parallelism abstractions. As with MULTILISP [Halstead 85] from which Butterfly LISP is derived, processor scheduling is handled through a global task queue. Tasks are entered in the queue with the `future` construct, which schedules the evaluation of a LISP form for parallel execution. In essence, the expression

```
(future form)
```

invokes a lazy evaluation of `form` that is dispatched through the queue. This construct is elegant, but potentially costly, since it requires dispatching the environment in which a form is evaluated along with the form.

Synchronization is achieved through a host of atomic test-and-set operations and locking primitives. For the purpose of this work, the synchronization primitives were unified by implementing atomic extensions to COMMONLISP's generalized variable access functions. Of particular interest is `setf-if-eq?!`. The form

```
(setf-if-eq?! place new old),
```

which compares the value of the generalized variable `place` to `old`, and if the test succeeds, atomically sets `place` to `new`. If the test fails, the variable is left unchanged, and the `setf-if-eq?!` returns `#!false`. Similarly, `atomic-incf!` and `atomic-decf!` respectively increment and decrement a generalized variable.

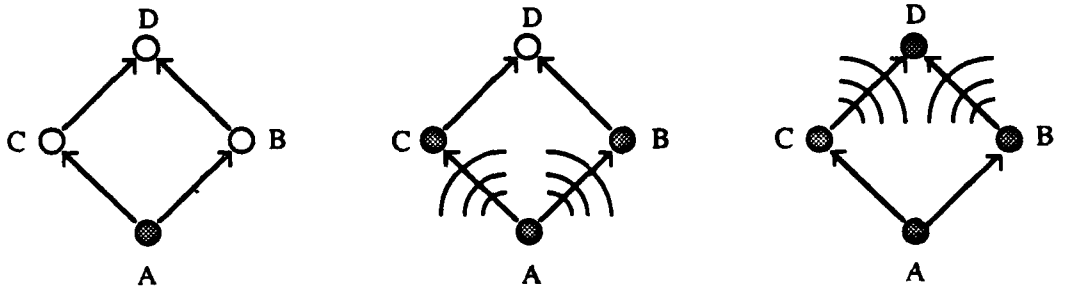
Finally, in Butterfly LISP, global variables are randomly allocated across all processors. Although this simple strategy does not allow user optimization of memory-to-processor allocation, it also minimizes memory contention by avoiding the concentration of variables on a particular processor's local memory. Empirical studies have demonstrated that this strategy leads to better performance for many problems than other seemingly more sophisticated approaches.

2.4 A parallel TMS algorithm

It is fairly clear, at least initially, where the architecture of the Butterfly and the parallelism constructs of Butterfly LISP might be exploited in a multiprocessing TMS. Since the machine has completely shared memory, the TMS constraint graph can be distributed across

all the processors: in essence this provides a global data structure for the graph, to which each processor has independent access within the same average time. Concurrency is introduced to the TMS in the main iteration step of the constraint propagation algorithm. In this step, the constraints attached to a node (e.g., clauses) are checked for potential deductions when a truth value is determined for that node (e.g., *true* or *false*). The concurrency is provided by checking these clauses in parallel, applying the parallel propagation recursively to any node whose truth is determined in the propagation process.

Figure 2: Concurrent Deductions



Assert A,
Constraints:
(not A) or B
(not A) or C
(not B) or D
(not C) or D

Propagate from A, concurrently inferring B and C

Propagate from B and C, yielding two support paths for D, but only inferring D once

This simple process can proceed without concern for synchronization, excepting for problems that arise when two processes attempt to set the truth of a node simultaneously. This situation can occur when each of several clauses that are being checked concurrently supports the determination of the same node (see Figure 2). Allowing the node's truth to be set repeatedly could lead to redundant propagation out of the multiply-deduced node, since each process that sets the node might try to propagate from it recursively. What is more, if the multiple clauses disagree as to the truth value that should be assigned to the

node, they might overwrite the truth setting of the node with conflicting values, leading to undetected contradictions.

This synchronization of propagation is obtained by atomically testing that the truth of a node is unknown before setting it. The synchronization thus occurs at the level of the Butterfly memory bus, which remains locked during the atomic test-and-set to all but one of the processes competing to set the truth of the node. Figure 3 shows a simplified LISP algorithm implements this synchronizing propagation.

This algorithm is used as follows. Given a database of nodes and constraints (i.e., non-unit clauses), a new proposition (i.e., a unit clause) is added to the database by calling `set-truth` on its corresponding node. The algorithm then performs truth maintenance over the database by computing all unit clause resolutions that the database sanctions. The correctness of the algorithm is readily verified.

Proposition 1: The algorithm in Figure 3 computes exactly those unit clause resolutions sanctioned by a database of asserted propositions and constraints.

Proof: First note that the algorithm never performs a resolution that isn't sanctioned by the database. Resolutions are performed in line 7, when `set-truth` is called to assign the truth of a node. However, `set-truth` is only passed such a node if this node is mentioned in a literal returned by `deducible-lit?` on line 5, which in turn happens just in case the literal is the only satisfied literal in a clause. Since all other literals in the clause are unsatisfied, inferring the literal exactly performs a unit clause resolution.

To see that the algorithm does not miss any resolutions, observe that a resolution must occur between a clause of length $n > 1$ and one or more unit clauses. However, a unit clause can only be added to the database through set-truth, which then checks all non-unit clauses that mention the unit clause in a literal (by a queued call to `propagate`, which in turn calls `deducible-lit?`). Thus, for any non-unit clause c, when enough unit clauses accumulate in the database to permit the resolution of c, the last such addition will cause `deducible-lit?` to sanction the resolution step on c.

Figure 3: Simplified Propagation Code

```
(define-struct node
  (truth 'unknown)  ; The truth status of a node.
  (clauses nil))    ; All clauses that mention this node.

1  (define (set-truth node truth)
2    (when (setf-if-eq?! (truth node) truth 'unknown)
     ; We set the node's truth.  Now propagate.
3      (mapc  (lambda (clause) (future (propagate clause)))
           (clauses node))))

4  (define (propagate clause)
5    (let ((literal (deducible-lit? clause)))
6      (when literal
       ; We found a literal (a node and its desired truth) to deduce.
7        (unless (set-truth (lit-node literal) (lit-truth literal))
         ; Another process set the node.  Check for contradictions.
8          (unless (eq? (truth (lit-node literal)) (lit-truth literal))
9            (flag-contradiction clause))))))

10 (define (deducible-lit? clause)
   (when (eq? (satisfied-literal-count clause) 1)
     ; The clause contains exactly one satisfied literal.  This
     ; literal mentions a node that is either unknown, or has truth
     ; value opposite to that which is assigned it in the literal.
     (get-satisfied-literal clause)))
```

Note that the preceding proposition only guarantees the overall correctness of the algorithm in the absence of undetected contradictions between assertions in the database. The synchronizations necessary to prevent contradictions from poisoning the database are guaranteed by proposition 2.

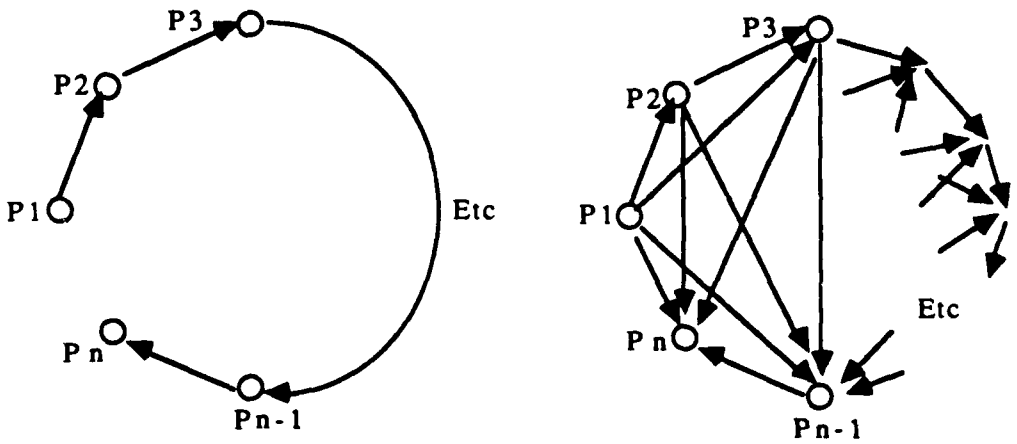
Proposition 2: The algorithm in Figure 3 correctly synchronizes parallel propagation in that (1) it prevents a node's truth from being set more than once, and (2) it detects any contradictions explicit in the database.

Proof: (1) The only way for a node's truth to be changed from its initial setting of *unknown* is through the `setf-if-eq?!` in `set-truth`. This atomic operation succeeds only when no other process has either previously set, or is currently setting, the node's truth. Since success is required to propagate the node's truth through its clauses (with the `mapc` in `set-truth`), only the first process that tries to set the node will spawn propagations.

(2) Conversely, an explicit contradiction arises in the database only when two clauses independently constrain a node's truth setting to have opposite values. However, the `unless` test in line 7 ensures that only one clause will succeed in propagating a value. An attempt to propagate to the node a second value, as assigned by a second clause, will fail the test in line 7, and the contradiction handler will then be invoked (lines 8 and 9).

It should be noted that the contradictions detected in this way must be explicitly present in the TMS. However, only when the TMS database is restricted to Horn and NoGood clauses will this be the case. In the general case, the incompleteness of constraint propagation as a decision procedure will allow some contradictions to go undetected.

Figure 4: Constraint Sequentiality



(a) inherent *(b) "compiled out"*

2.5 Defeating the Sequentiality of Truth Maintenance

As noted above, the P-completeness of unit clause resolution guarantees that some truth-maintenance problems are inherently non-parallelizable, and must be solved sequentially. Figure 4a shows a TMS constraint graph of this type. In this example, if *P1* is asserted, the entire graph will have to be traversed in sequence before all deductions are completed.

In practice constraint sequentiality is not necessarily this pathological. When it has occurred in the circuit simulations that have served me as examples, the main source of sequentiality was a coupling of moderate constraint fanout (of order 2 or 3) with a predominance of constraints that had a low probability of leading to an actual deduction when checked. The result was short bursts of moderate parallel activity over a background of uniprocessing.

It is a folk truism, however, that this kind of run-time sequentiality can be eliminated through compilation. In theory, one can short cut a series of n constraints that are all satisfied by the same premise set by compiling a new constraint for each of the additional $n(n+1)/2$ subsets of the series. The graph in Figure 4a would thus be transformed into that of Figure 4b. At run time, adding the premises that satisfy any one of the constraints in the original series would now require $O(n)$ parallel propagation steps to transmit the premises through the series. However, each propagation step requires a logarithmic parallel time factor in order to map through the constraints indexed to the node (if the constraint sets are optimally organized as balanced trees). This in turn translates to $O(log\ n)$ parallel time to transmit the premises through the compiled graph for the series, as opposed to $O(n)$ parallel time for the original graph.

Note that this doesn't eliminate the inherent sequentiality of the problem, since $O(n)$ parallel time or $O(n^2)$ serial time is required to compile the graph. Additionally, the $O(log\ n)$ parallel run time performance requires $M(n^2)$ processors, since each of the compiled series' $n(n+1)/2$ constraints may need to be checked at run time.

2.6 Additional Concurrency Strategies

Although graph compilation can increase the degree of run-time concurrency for inherently sequential TMS databases, the resulting $M(n^2)$ processor requirement is unrealistic for all but the most massively parallel of multiprocessors. For smaller machines, especially MIMD ones like the Butterfly, other sources of run-time concurrency must be exploited in order to maximize processor utilization.

One way to increase the concurrency of the TMS is to assert multiple premises simultaneously. To do this, one simultaneously labels all the nodes corresponding to the premises with their respective truth settings, and allows constraint propagation to proceed in parallel from each of the premises. The result is a kind of pipelining in which separate streams of constraint propagation occur in parallel with near independence. Even if each independent assertion would only lead to moderate processor utilization, asserting them together increases the degree of useful parallelism. The limiting factor thus becomes the degree to which multiple premise assertions need be made at once.

In fact, in this model of truth maintenance, simultaneous premise assertion arise naturally when switching contexts to explore rival hypotheses. To avoid the large potential overhead of maintaining many contexts simultaneously [Provan 88], only one global context is maintained by the TMS at any one time. Switching contexts therefore requires a series of assertions and retractions, which can be performed concurrently in this TMS.

Retraction is another source of TMS concurrency. Algorithmically, retracting a premise is analogous to asserting one, with the removal of truth settings replacing their addition. As a process, retraction shares the same properties as assertion, including worst-case sequentiality and the opportunity to pipeline several retractions simultaneously. However,

retraction may not safely co-occur with assertion. Indeed, consider a circular constraint chain. If a node is asserted by one process, and then immediately retracted by another, the two processes could loop around the chain repeatedly (possibly indefinitely), the one process removing what the other has just added.

Finally, concurrent strategies also exist for constructing the constraint graph. It is possible to add multiple constraints to the graph at the same time, provided some, by now familiar, synchronization concerns are addressed. For example, indexing a constraint on a node's constraint list must be done atomically, to ensure well-formedness of the list if several constraints are being indexed concurrently.

More significantly, the truth maintenance algorithm of Figure 3 can be extended to allow construction of the constraint graph to occur simultaneously with constraint propagation. That is, portions of the constraint graph can be under construction while premise assertion and deduction are occurring. This is somewhat counter-intuitive, since it can lead to some processors modifying graph structures at the same time as these structures are being explored by other processors. The synchronization of these disparate processes is achieved by explicitly locking a TMS node at precisely those times when graph construction and deduction would adversely interact. These locks occur at the software level, and unlike the atomic memory bus locks that introduce virtually no delay, these software locks may force processes to go to sleep temporarily, i.e., be rescheduled for subsequent re-evaluation. Fortunately, the locks, are only in effect for brief periods of time, and don't seem to involve many processes beyond their initiator.

The algorithmic minutiæ of this locking strategy are not of immediate interest here, however, and have been relegated to the Appendix. What *is* of interest is the fact that

allowing assertion, deduction and graph construction to occur in parallel endows the TMS with a broad heterogeneous concurrency.

2.7 Heterogeneous Concurrency

This heterogeneous concurrency has an unexpected consequence on the way one approaches truth maintenance. Indeed, it is now possible to view the entire process of building a model in the TMS as a naturally parallel one. Current (uniprocessor) TMS's have cast this modeling into explicitly separate phases of problem solving (in which constraints get added to the graph) and hypothesis testing (in which premises are asserted and then propagated). On a uniprocessor, this is a natural distinction to draw, since computation must necessarily be serialized, and may as well be so in a way convenient to the programmer. On a MIMD multiprocessor, not only is serializing problem-solving and hypothesis testing unnecessary, but it is to be avoided whenever it reduces the degree to which concurrency can be exploited. To the extent that a problem supports it, the TMS programmer should be free to mix problem-solving with truth maintenance.

One way to enable this naturally is through noticers (demons) placed on TMS nodes. Almost all TMS's provide these kinds of demons to inform the problem solver of a salient proposition becoming believed or disbelieved as a result of deduction. In a MIMD environment like that of the Butterfly multiprocessor, these demons can be run concurrently with the TMS, as can the problem-solving processes the demons in turn spawn. Although doing so promotes concurrency, it also raises an issue of consistency.

Indeed, the premises that lead to a demon being fired might ultimately be contradicted by other premises added at the same time. However, since truth maintenance could still be

occurring at the time of the demon's invocation, the contradiction may not yet have been detected. Hence the problem solver may not unconditionally assume that the premises that led to its being invoked are not at risk of being contradicted. The burden is on the problem solver to determine at some point in the future whether such a contradiction occurred.

How best to handle the concurrent interaction of the TMS and the problem solver is very much an open question. The current implementation of the system doesn't address this issue beyond providing a framework in which it can be explored.

2.8 Performance Results

All of the algorithms and strategies described in this paper have been implemented in a running Butterfly LISP program. This TMS has been put to use in symbolic simulation of digital circuits, an application in which it has demonstrated significant concurrency. Even the simulation of simple circuits, such as a 4 bit adder, can lead to the full utilization of the 16 processors on the machine that I have used. This is the good news.

The not-so-good news is that the actual running time of this multiprocessing TMS can actually be worse than that of a uniprocessor system[4] running on the same problem. The reason for this disappointing performance is, I think, largely due to the extreme youth of the underlying LISP system. Indeed, at the time the TMS was first implemented, no compiler was available for Butterfly LISP, and much of the system code itself was running interpreted. A compiler has recently been made available, and I am in the process of tuning the TMS to the new LISP environment, with the expectation that this will lead to more rewarding performance measurements!

[4] A Symbolics 3670, to be precise.

Additionally, until the TMS can be run compiled, it is hard to evaluate the degree to which the concurrency it displays represents effective computation. The concurrency would be uninteresting if it only amounted to, for example, mindless enqueueing and dequeueing of processes that performed no useful work before being put back to sleep. Ineffectiveness of this sort does not currently seem to be occurring. However, this conclusion must remain tentative until the system is tested in a more realistic computational environment.

Along these lines, the `future`-based task queue appears to be too general for the needs of the TMS. In fact, all of the parallel mapping operations in the TMS can be recast in terms that do not require packaging up lexical environments, as is done with `future` forms. One way to optimize the TMS would thus be to circumvent the system's task queue in favor of one without the `future` overhead. Several optimizations such as this will likely be necessary before timing statistics can be fairly obtained for this system, and before comparisons can be accurately drawn to other implementations on other hardware.

2.9 Cross-applicability of These Results

It is enlightening to consider these results in view of how they might apply to other TMS models, and to other parallel hardware. De Kleer's ATMS [de Kleer 86], for example, extends McAllester's RUP with multiple simultaneous contexts and a minimization operation on premises. This system has found uses, particularly in solving problems in which all solutions are required; for such problems the ATMS provides efficient sharing of intermediate results. Although this is still an open issue, it is not immediately apparent how to implement the sharing and minimization in parallel without requiring costly global synchronizations. Without these features (but with contexts), a simplified, though less

interesting ATMS could readily be made concurrent with the techniques described here. Additionally, this restricted ATMS could also be run in concert with its problem solver.

It is reasonable to wonder whether the parallel techniques described here can be generalized to all constraint propagation problems; to some extent they can. As long as the problem's constraint formalism maintains some analog of deduction (numeric assignment for example), the concurrent graph building and propagation strategies are applicable. Also, if the problem supports the propagator/solver architecture used here, the analogous components may be run concurrently too.

Another open question is whether the sources of parallelism that have emerged in this MIMD TMS can also be exploited on large-scale SIMD machines such as the Connection Machine™ [Hillis 85]. This seems somewhat unpromising, since given the worst-case sequentiality of constraint propagation, the current implementation achieves much of its parallelism through the *heterogeneity* of its concurrent tasks. It isn't clear how this heterogeneity can be orchestrated on machines whose processors operate solely in locked synchrony. This isn't to say that SIMD truth maintenance systems are unfeasible. For example, an intriguing ATMS variant has been implemented on the Connection Machine [Dixon & de Kleer 88]). Instead of sharing deduction, or computation of any sort, between contexts, this system places one context on each of the machine's 65,536 processors, and performs a *serial* truth maintenance problem in each context.[5] What systems such as this demonstrate is that SIMD truth maintenance systems almost certainly require a different set of techniques than those I've described here.

[5]The enormity of this computation demonstrates graphically the actual cost of determining the 2^n solutions to a problem that follow from considering all possible combinations of n Boolean hypotheses. The entire Connection Machine is taken up in computing the consequences of only 16 hypotheses.

Indeed, one of the lessons of this work has been that the sources of parallelism inherent in a problem are often not apparent until one tries to elicit them. Further, the parallel strategies that one discovers in this way are necessarily optimal for the class of machine at hand. These hardware-imposed constraints should not, however, be seen as limitations. Only by exploring the full range of models of concurrency will we come to a full understanding of the impact of parallelism on theories of intelligence.

References

de Kleer, J. (1986). An Assumption-Based Truth Maintenance System. *Artificial Intelligence* **28**, pp. 127-162.

Dixon, M. & de Kleer, J., (1988). Massively Parallel Assumption-Based Truth Maintenance. In *Proceedings of AAAI-88*.

Doyle, J. (1978). *Truth Maintenance Systems for Problem Solving* (AI TR 419). MIT Artificial Intelligence Lab.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Co.

Halstead, R. H., Jr. (1985). Multilisp: A Language for Concurrent Symbolic Computation. *ACM Trans. on Programming Languages and Systems* **7**, pp. 501-538.

Hillis, W. D. (1985). *The Connection Machine*. Cambridge, Ma: The MIT Press.

Jones, N. D. and Laaser, W. T., (1977). Complete Problems for Determi-nistic Polynomial Time. *Theoretical Computer Science 3*, pp. 105-117.

Kasif, S. (1986). On the Parallel Complexity of Some Constraint Satisfaction Problems. In *Proceedings of AAAI-86* , pp. 349-353.

McAllester, D. A. (1980). *An outlook on Truth Maintenance* (AI Memo 551). MIT Artificial Intelligence Lab.

McAllester, D. A., (1984). *A Widely Used Truth Maintenance System.* Unpublished technical report, MIT Artificial Intelligence Lab.

Provan, G., (1987). Efficiency Analysis of Multiple-Context TMSs in Scene Representation. In *Proceedings of AAAI-87*, pp. 173-177.

Reif, J., (1985). Depth-First Search is Inherently Sequential. *Information Processing Letters 20*, pp. 229-234.

Steinberg, S. A., Allen, D., Bagnall, L., Scott, C. (1986). The Butterfly Lisp System. In *Proceedings of AAAI-86*, pp. 730-734.

Waltz D. (1975). Understanding Line Drawings of Scenes with Shadows. In Winston, P. H., editor, *The Psychology of Computer Vision.* New York: McGraw-Hill, pp. 19-92.

Zabih, R., McAllester, D. A., and Chapman, D., (1987). Non-Deterministic Lisp with Dependency-Directed Backtracking. In *Proceedings of AAAI-87*, pp. 59-64.

Butterfly™ is a trademark of Bolt Beranek and Newman, Inc.
Connection Machine™ is a trademark of Thinking Machines Corporation.

Appendix: Heterogeneous Propagation Algorithm

For the truly inquisitive, this appendix contains a barely simplified version of the fully heterogeneous truth maintenance algorithm. This algorithm allows for simultaneous constraint propagation and constraint definition (graph construction). The key to enabling these two disparate processes to co-occur lies in the proper synchronization of two specific steps: (1) the setting of the truth of a TMS node, and (2) the indexing of a constraint to a node. This synchronization is achieved by explicitly locking a node whose truth is unknown before attempting either of these steps. The node remains locked just long enough for the step to take place.

For the truth-setting step, shown below in Figure 5, the locking amounts to setting the truth of the node to `'locked`, saving in a temporary register the clauses currently indexed to the node, and resetting the truth to the originally desired value (lines 1', 2', and 3' in the Figure). The clauses saved in the temporary register are exactly those that "think" the node has truth unknown.[6] After the node's truth is reset in line 3', these saved clauses must be propagated. If the node were not locked in this way, a clause initialized when the node was still unknown might possibly be indexed to the node between the time the node's truth was set and the time the node's clauses were consequently inspected for propagation (i.e., between lines 2 and 3 of the algorithm given earlier in Figure 3). This furtive clause would then have its `satisfied-lit-count` slot set to a value inconsistent with the state of the TMS.

[6]Indeed, in their `satisfied-lit-count` slot, clauses cache a count of their satisfied literals. If a node has been assigned the truth specified by a literal, or remains unknown, then the literal is considered satisfied.

The code for the clause indexing step, shown below in Figure 6, capitalizes on the temporary locking provided by the truth-setting step. When `index-clause` is called to add a clause to the graph, it tries to lock all the currently unknown nodes mentioned by the clause (the `mapc` started on line 6' in the Figure). If in this process a node becomes locked by `set-truth` (or by another `index-clause` process), `index-clause` aborts and requeues itself for future evaluation (line 10'). If, however, the unknown nodes are successfully locked (`if` of line 7'), the clause can have its `satisfied-lit-count` slot properly initialized (line 9'), and can be indexed to the requisite nodes (line 8'). Additionally, since the unknown nodes are temporarily locked during this time, no `set-truth` process can surreptitiously set the truths of the nodes and invalidate the clause's `satisfied-lit-count`.

One final wrinkle: if a set-truth process attempts to set a node temporarily locked by an index--clause or another set-truth process, the first set-truth process may have to be put to sleep. This is handled in lines 4' and 5' of Figure 5.

Figure 5: Heterogeneous Truth-Setting Code

```
(define-struct node
  (truth 'unknown)  ; The truth status of a node.
  (clauses nil))    ; All clauses that mention this node.

(define-struct clause
  (literals nil)              ; The literals that make up the clause,
                              ; sorted by a canonical order on nodes.
  (satisfied-lit-count 0)) ; How many of the literals are satisfied?

(define (set-truth node truth)
1'   (if (setf-if-eq?! (truth node) 'locked 'unknown)
     ; The node is now locked:  no clauses can be added.
2'     (let ((clauses (clauses node)))
       ; Temporarily save the clauses that think this node is unknown.
3'       (setf-if-eq?! (truth node) truth)
       ; We set the node's truth and unlocked the node.  Can add
       ; clauses safely now.  As before, propagate
       (mapc  (lambda (clause) (future (propagate clause)))
              clauses))
     ;; If we failed the setf-if-rq?!, it could be because the node was
     ;; locked.  If it was, or is still unknown, try again.
4'     (when (locked-or-unknown? (truth node))
5'       (future (set-truth node truth)))))

(define (propagate clause)
  (atomic-decf! (satisfied-lit-count clause))
  (when (eq? (satisfied-lit-count clause) 1)
    (let ((literal (get-satisfied-literal clause)))
      ; We found a literal (a node and its desired truth) to deduce.
      (unless (set-truth (lit-node literal) (lit-truth literal))
        ; Another process set the node.  Check for contradictions.
        (unless (eq? (truth (lit-node literal)) (lit-truth literal))
          (flag-contradiction clause))))))

(define (get-satisfied-literal clause)
  ; Extracts the one satisfied literal in this clause.
  ...)
```

Figure 6: Heterogeneous Constraint-Indexing Code

```
(define (satisfied-lit? literal)
  ; The node in the literal has the truth required by the literal.
  (or  (eq? truth (lit-node literal)) 'unknown)
       (eq? (truth (lit-node literal)) (list-truth literal))))

(define (index-clause clause)
  ; Indexes a clause in the constraint graph.
  (let ( (succesful-lock? #!true)
         (locked-node nil)
         (sat-count 0))
6'       (mapc (lambda (literal)
                 ; Map over unknown nodes of clause, and try to lock them.
                 (let ((node (lit-node literal)))
                   (when (eq? (truth node) 'unknown)
                     (if (setf-if-eq?! (truth node) 'locked 'unknown)
                       (push! node locked-nodes)
                       (setq! succesful-lock? #!false)))))
               (literals clause))
7'       (if succesful-lock?
         (sequence
           ; Locked all unknown nodes in clause.  Now, index the clause.
8'           (mapc (lambda (literal)
                    (atomic-push! clause (clauses (lit-node literal)))
                    (when (satisfied-lit? literal) (incf! sat-count)))
                  (literals clause))
           ; Initialize the clause, and unlock the nodes.
9'           (setf! (satisfied-lit-count clause) sat-count)
           (mapc  (lambda (node) (setf! (truth node) 'unknown))
                  locked-nodes)
           ; Propagate the clause, if need be.
           (future (propagate clause)))
       (sequence
           ; Didn't succeed in locking the necesary nodes.  Go to sleep
           ; and try later.  But first restore locked nodes.
           (mapc  (lambda (node) (setf! (truth node) 'unknown))
                  locked-nodes)
10'          (future (index-clause clause))))))
```

3. Sentential Semantics for Propositional Attitudes

Andrew Haas, BBN Systems and Technologies Corporation

3.1 The Notion of a Thought Language

The sentential theory of propositional attitudes claims that propositions are sentences of a thought language. It has an obvious appeal to AI workers, since their programs often contain sentences of an artificial language, which are supposed to represent the program's beliefs. These sentences can be true or false, and an agent can make inferences from them, so they have two essential properties of beliefs. It is tempting to conclude that they are the program's beliefs, and that human beliefs are also sentences of a thought language. If we extend this to all propositional attitudes, we have a sentential theory of propositional attitudes. Such a theory has an advantage over a possible worlds theory because it does not imply deductive closure. Indeed, it does not imply anything about what agents will deduce from their beliefs. It simply leaves this question open.

Montague used the possible worlds theory to construct a semantics for a fragment of English, including propositional attitudes (Montague 1970). In this fragment he addressed the key problem of quantifiers that are outside the scope of a propositional attitude, but which bind variables inside that scope. This phenomenon is briefly known as "quantifying in". An example is "There is someone that Ralph believes to be a spy". Konolige (1986) treated this problem within a sentential theory, but he assumed that each object in the

domain of discourse has a unique standard designator in the thought language. As Moore (1988) observed, this is very implausible. This paper will present a semantics for a fragment of English based on a sentential theory. This fragment includes quantifying in, but it does not require unique standard designators.

Moore (op. cit.) states that such fragments "invariably become very complicated, and seem to require a treatment of quantification into attitude reports that is quite different from the treatment of other types of quantification" (p. 8). We leave it to the reader to judge whether our fragment is too complicated. It is certainly true that we need special mechanisms to handle quantification into the scope of attitudes, but every theory on the market is forced to admit that attitude reports have unique properties. The situation semanticists claim that they have a unique pragmatics, while Cresswell says that they have a unique kind of ambiguity. The real question is which kind of uniqueness best explains the facts.

As AI researchers, we seek to explain facts about ordinary human behavior. This involves analyzing an ordinary person's tacit understanding of propositional attitudes, and turning that analysis into a program. This common-sense understanding is probably inadequate for psychology and philosophy - just as the common sense view of space and time is inadequate for physics (see Churchland, 1979). Thus the sentential theory may be false, but perfectly adequate for describing the inferences about belief and knowledge that people make in daily life. An argument against this theory is relevant to the present paper only if it shows that the theory cannot reproduce common-sense inferences about attitudes.

Here is an example of an argument against the sentential theory that is plausible, but irrelevant to our goals. Moore (op. cit., p. 8) argues that dogs have beliefs, but they

cannot have a thought language like human thought language, so beliefs are not sentences of thought language. We reply that people try to explain the actions of dogs by assuming that they use a very limited subset of human thought language. Quite likely this is wrong - which explains why we don't understand the actions of dogs nearly as well as the actions of people. Finding a good theory about the actions of dogs is an important problem in psychology. *It is irrelevant to this paper - just as quantum mechanics is irrelevant to* Hayes's program of naive physics.

To make this distinction clearer, we cite an argument against the sentential theory that is relevant to our concerns. It comes from Levesque (1984). Consider the following inference:

> Mary said that Bill is a fool and Tom is an idiot.
>
> Sue said that Tom is an idiot and Bill is a fool.
>
> ---
>
> Sue said the same thing that Mary said.

This is a valid inference. To put it in AI terms, a program that accepts the premisses and rejects the conclusion would fail the Turing test. Yet a standard version of the sentential theory would represent the inference as follows:

> say(mary, "fool(bill) $\wedge$ idiot(tom)")
>
> say(sue, "idiot(tom) $\wedge$ fool(bill)")
>
> ---
>
> ($\exists$ x say(sue, x) $\wedge$ say(mary, x))

This inference is invalid, because the sentences "fool(bill) ∧ idiot(tom)" and "idiot(tom) ∧ fool(bill)" are not identical. The sentential theory is making overly fine distinctions - it claims that two sentences express different propositions when the difference is merely in the word order.

Konolige (1986) offered a simple correction to the sentential theory that solves this problem. Suppose that sentences of thought language are ordered, labeled trees - that is, the children of each node are totally ordered. Then (P ∧ Q) and (Q ∧ P) are two different trees. But let us suppose instead that some nodes in our trees have an unordered set of children. In particular, if a node bears the label ∧ or ∨ its children are unordered, while if it bears the label → its children are ordered. If this is true, then (P ∧ Q) and (Q ∧ P) are two different notations for one tree, and the inference cited above becomes valid. We will not incorporate this idea into our fragment, since we wish to concentrate on the problem of quantifying in. Still the fragment could easily be extended to allow operators with unordered sets of arguments. Thus we reject the goal of basing a semantics for English on a true theory about propositional attitudes. However, we must replace this requirement with another requirement peculiar to AI. Our ultimate goal is to build a robot that can use English to talk about both human propositional attitudes and its own propositional attitudes. The most obvious way to do this is to give the robot one theory of propositional attitudes, which it can apply to itself and to human beings as well. Therefore we insist that it must be possible to build a robot which exemplifies our theory about thought language. We are not allowed to make assumptions that are clearly false for any feasible robot - for instance, the assumption that the agent knows every logical consequence of its knowledge.

We imagine that our agents are robots guided by AI programs of a familiar kind. The agents think by manipulating sentences of a logical language, in the familiar fashion. These

sentences are labeled trees, and the labels are atomic symbols. As always in computer science, a single mathematical object can have different representations in different programs. In a typical program, atoms of LISP might represent atomic symbols of thought language. However, we do not identify LISP atoms with symbols of thought language. Different robots can use different LISP atoms to represent the same symbol of thought language. It is up to us to stipulate that the atom C used by robot X and the atom D used by robot Y represent the same thought-language symbol. For example, if robot X uses atom C to represent the concept of a dog, and Y uses D to represent the same concept, we may stipulate that C and D are both representations for the same abstract symbol.

We assume that each agent has sensors, which continuously create new beliefs about the agent's surroundings. Each agent has effectors, which accept descriptions of actions in thought language and attempt to execute those actions. Agents can form new beliefs by deduction and by introspection. A sentential theory gives us a great deal of freedom in describing what inferences an agent will make given certain beliefs and goals. Haas (1986) offers some proposals about these questions, while Konolige (1986) offers a different approach. For present purposes it is enough to describe the effects and preconditions of deduction and introspection, without specifying which deductions or introspections the agent will perform.

Deduction is straightforward - it creates new beliefs which follow from old ones according to certain proof rules. Introspection is more difficult. Suppose an agent learns by introspection that he believes the sentence "white(snow)". Then he has a new belief which asserts that he believes "white(snow)". This belief must contain some name for the agent himself - but what name? To answer this question we assign each agent a special name in

thought language called his selfname. Suppose John's selfname is "Me_1". Then if John learns by introspection that he believes "white(snow)", the new belief is

believe(Me_1,"white(snow)")

If there is a table in front of John, his sensors will produce the input sentence

$(\exists\ x\ \text{table}(x) \wedge \text{in-front}(x,Me_1))$

If he wants to achieve a goal sentence G, he tries to find a plan that he can execute in order to make G true. That is, he tries to prove a sentence of the form

$\text{execute}(Me_1, \text{plan}_1) \rightarrow G$

If he succeeds in proving this theorem with $\text{plan}_1 = P$, he will execute the plan P.

The constant Me_1 is important because the sensors, the introspection mechanism, and the planning mechanism all give that constant special treatment. Notice that if the agent in question is a robot, we can make these three statements true by straightforward programming. For example, the introspection program will accept a sentence p and try to decide if it is among the program's beliefs. If it is, the program will create a new belief:

believe(Me_1,"p")

The selfname will play an important role in our semantics, and its unique properties are procedural. This means that we do not have a purely model-theoretic semantics for natural language - we incorporate procedural ideas.

3.2 Vivid Designators

De re belief reports are a key problem in the semantics of propositional attitudes. A pleasant example comes from a text on ancient history: "Hellenistic scholars believed that Troy fell in 1184 B.C." The Hellenistic period ended before Jesus Christ was born, so it is clear that these scholars did not believe Troy fell 1184 years before the birth of Christ. They referred to the same year by a different description - in fact they counted from the first Olympiad, in 776 B.C. of our calendar. Thus the noun phrase "1184 B.C." denotes the year that Hellenistic scholars believed Troy fell in, but it gives us no clue about how they described that year.

This creates a particular difficulty for the sentential theory. If beliefs are sentences, then a belief about the year 1184 B.C. must contain some expression of thought language that denotes the year 1184 B.C. So which expression? One can imagine a variety of them, but the belief report by itself gives no clue about which one was in the minds of the Hellenistic scholars. This is not a difficulty for the possible worlds theorists, since they analyze beliefs without any appeal to mental descriptions.

One possibility is that the sentence simply asserts that the Hellenistic scholars had some mental description X of the year 1184 B.C., and they thought Troy fell in the year X. This seems too weak, however. Suppose that Agamemnon was murdered in 1184 B.C., and the scholars believed that Troy fell in the same year Agamemnon was murdered. Suppose further that they agreed this year was 400 years before the first Olympiad - that is, the year

1176 B.C. of our calendar. If these are the facts, then they believed two sentences of thought language that looked something like this:

fell(troy,(ι y year(y) ∧ murdered(agamemnon,y)))

fell(troy,(ι y year(y) ∧ before(y,first-olympiad,400)))

Consider the second argument of the predicate "fell" in the first sentence. It is indeed a description of the year 1184 B.C., so the scholars believed Troy fell in year X, where X denotes 1184 B.C. Still nobody would say that if the scholars believed this sentence, they therefore believed that Troy fell in 1184 B.C. The thought language description

(ι y year(y) & murdered(agamemnon,y)))

is too weak to support a de re belief report. We would say instead that if they believed the second sentence, they believed Troy fell in 1176 B.C. The thought language description

(ι y year(y) & before(y,first-olympiad,400)))

is strong enough to support a de re belief report.

Kaplan (1969) described these problems, and he offered a solution. He distinguished a class of terms called vivid designators - those that support a de re belief report. Then the example sentence will mean that the Hellenistic scholars believed that Troy fell in the year X, where X is some vivid designator. Kaplan's vivid designators were Fregean concepts, while ours are closed terms of thought language. This thought language is a theoretical entity, and we assign it the properties we need. In particular, we are free to add new kinds

of vivid designators to the thought language in order to create the de re readings we need. These vivid designators might not have any translations into English. For example, when we perceive an object we create a vivid designator for that object, but that vivid designator has no obvious English translation.

The notion of a vivid designator is also relevant to wh-questions. When I ask you "Where is X?", I am requesting a response that will help me to find a vivid designator for the location of X. Yet the question seems to be more specific than this. In some examples there is an intuition that the speaker has a particular kind of vivid designator in mind. Suppose you and John are staying in a hotel in a strange city and you go out for a walk. After a while John asks "Do you know where we are?" You realize that you're completely lost, and reply "No". Seeing a telephone you decide to call Mary and ask for directions. She answers and says "Do you know where John is? I have to talk to him right away". You answer "Yes, he's right here", and hand him the phone. When John asked if you knew where he was you said no; a minute later you said yes. Presumably you understood that the answer "here", although it allows John to find a vivid designator for his location, is not helpful because he already has that vivid designator. In examples like this the speaker wants a particular kind of vivid designator, and he expects the hearer to understand which kind. To capture this intuition we postulate an ambiguity in sentences that involve quantifying in. If an English sentence says that John has a belief about an object x, its logical translation will indicate what kind of vivid designator of x appears in John's belief. That means that there are as many translations as there are kinds of vivid designators. This will not necessarily create a computational problem, because a program may not have to enumerate the possibilities in order to choose the right one.

Unfortunately this paper will say little about ways to discover from context what kind of vivid designator a speaker has in mind. To study this problem one must choose a domain and describe it in detail - including the knowledge preconditions of various actions. This paper presents a formal semantics for a fragment of English, which generates the set of logical translations that the hearer must choose from.

3.3 The Thought Language of our Fragment

Our semantics will provide a translation from English sentences to sentences of thought language. This thought language is a first-order logic with minor extensions. In our previous discussion we said that sentences of thought language are labeled trees. In our fragment we will take a slightly different tack, regarding logical expressions as lists. The reason for our choice is that we are going to use pure Prolog as a notation for our syntax and semantics, and if expressions are lists we can take advantage of Prolog's convenient notation for lists. In this notation we write [a, b, c] instead of (cons 'a (cons 'b (cons 'c nil))), and we write [a | X] instead of (cons 'a x) (remember that Prolog variables begin with capital letters). When we write expressions of thought language we will take a slight liberty with Prolog's notation by omitting the commas between elements of a list.

The quantifiers of our language are more complicated than in a standard first-order logic. This is because we analyze noun phrases as complex quantifiers, as Montague did. In order to make translation as easy as possible, we include complex quantifiers in our logical language. Of course there is a price for this: in order to apply standard theorem-proving methods, we must replace the complex quantifiers with combinations of quantifiers and connectives. This, however, is easy to do.

We will therefore translate "John saw a cat and a dog" as

[[[∃ x [cat x]] ∧ [∃ x [dog x]] [see john x]]]

The expression

[[∃ x [cat x]] ∧ [∃ x [dog x]]

is a complex quantifier. It applies to a wff, say [p x], and returns true iff there is a value for x that makes [cat x] and [p x] true, and also a value for x that makes [dog x] and [p x] true.

A model consists of a set D called the domain of discourse and a function f that maps constants to elements of D and n-ary predicate letters to functions from D^n into the set {T,F}. The language contains no function letters. Let M be a model with domain D. Let den(t,M) be the denotation of a symbol or expression t in M. If *e* is an environment and *v* a variable, let *e*(*v*) be the value that *e* assigns to *v*. If d ∈ D, let *e*[*v* := d] be the environment like *e* except that it assigns d to *v*. Obviously *e*[*v* := d](v) = d.

Variables are the letters x,y,z, and w, possibly with subscripts. We write $<t_1...t_n>$ for the n-tuple of $t_1...t_n$. The following definitions should need no explanation:

Every variable v is a term, and den(v,e,M) = e(v).

Every constant c is a term, den(c,e,M) = f(c).

If *p* is a predicate letter of *n* arguments and $t_1...t_n$ are terms, then [*p* $t_1...t_n$] is a wff. den([*p* $t_1...t_n$],e,M) is T if <den(t_1,e,M)...den(t_n,e,M)> ∈ f(p), otherwise F.

If p and r are wffs [∧ p r] is a wff and den([∧ p r],e,M) = T if den(p,e,M) = T and den(p,e,M) = T, otherwise F.

If p and r are wffs [∨ p r] is a wff and den([∨ p r],e,M) = T if den(p,e,M) = T or den(p,e,M) = T, otherwise F.

The syntax of complex quantifiers is as follows. If v is a variable and p and r are wffs, then

[∃ x p r]

[∀ x p r]

[∃! x p r]

[~∃ x p r]

are quantifiers. If r and s are quantifiers, then

[r ∧ s]

[r ∨ s]

are quantifiers. Finally, if q is a quantifier and p a wff, [q p] is a wff.

In standard first-order logic, there is no need to assign a denotation to the symbol ∃ - it simply triggers a rule that assigns denotations to wffs of the form (∃ x p). In the same

way, the complex quantifiers have no denotations of their own. The semantics of quantified wffs is as follows:

den([[∃ x p],r],e,M) = T if for some d ∈ D, den(p,e[x := d],M) = T and den(r,e[x := d],M)= T, otherwise F.

den([[∃! x p],r],e,M) = T if for exactly one d ∈ D, den(p,e[x := d],M) = T and den(r,e[x := d],M)= T, otherwise F.

den([[~∃ x p],r],e,M) = T if for no d ∈ D, den(p,e[x := d],M) = T and den(r,e[x := d],M)= T, otherwise F.

den([[∀ x p],r],e,M) = T if for all d ∈ D, den(p,e[x := d],M) = F or den(r,e[x := d],M)= T, otherwise F.

These definitions are minor variants of the standard definitions. The definitions for conjunctive and disjunctive quantifiers are more novel:

den([[q_1 ∧ q_2] r],e,M) = T if den([q_1 r],e,M) = T and den([q_2 r],e,M) = T, otherwise F.

den([[q_1 ∨ q_2] r],e,M) = T if den([q_1 r],e,M) = T or den([q_2 r],e,M) = T, otherwise F.

Notice that [q_1 r] is not a sub-expression of [[q_1 ∧ q_2] r]. Therefore the truth value of [[q_1 ∧ q_2] r] is not defined in terms of the truth values of its sub-expressions, as in the standard treatment of first-order logic. However the recursive definition is clearly correct, because

the number of symbols in the expressions decreases at each recursive step. The above definitions obviously imply that [[$q_1 \wedge q2$] r] is true iff [[q_1 r] $\wedge$ [q_2 r]] is true, and [[$q_1 \vee$ $q2$] r] is true iff [[q_1 r] $\vee$ [q_2 r]] is true. Thus it is easy to eliminate the conjunctive and disjunctive quantifiers after they have served their purpose of simplifying the translation rules.

The definition of free and bound variables must be extended to allow for the complex quantifiers. The quantifier [$q_1 \wedge q2$] binds a variable v iff q_1 and q_2 both bind v,and similarly for [$q_1 \vee q2$]. If q is a quantifier and r a wff, the free variables of [q r] are all free variables of r except those bound by q. The reader can check that with this definition the denotation of a wff depends only on its free variables, as in standard first-order logic.

We need to extend our logic in another direction in order to represent propositional attitudes and quantifying in. Since propositions are wffs of thought language, the domain D must include every wff of the language. To include the expressions of the language in the domain of discourse is certainly unusual, but it is perfectly consistent with the standard definitions of first-order logic, which place no restrictions on the entities that may occur in the domain. It creates no self-reference paradoxes - these arise only when we try to add the predicate "true" to the language. No first-order language can contain its own Tarskian truth predicate, but it is quite possible for the language to contain a truth predicate that approximates the Tarskian truth predicate, disagreeing only in the paradoxical cases. Perlis (1985) has shown how this is done, and proved that the resulting systems are consistent. Haas (1986) also discusses the matter.

The language includes a set K of distinguished constants which denote the various kinds of vivid designators - numerals, selfnames, and whatever other kinds of vivid designators we

may need. An object may or may not have a vivid designator of a particular kind - for example, only numbers have arabic numerals, and only agents have selfnames. No object can have more than one vivid designator of a particular kind - each agent has just one selfname, and each number has just one arabic numeral. All vivid designators are closed terms of the thought language.

There is also a a special predicate "has-name". [has-name x n k] means that n is a name for object x and n belongs to the kind k of vivid designators. For example, if "23" denotes the arabic numeral for the number 23, we write

[has-name 23 "23" arabic]

If "Me$_1$" denotes the selfname of John, we write

[has-name John "Me$_1$" selfnames]

To put if formally: if k is the denotation of a constant in K, then for all $x \in D$ there is at most one $n \in D$ such that the triple $\langle x\ n\ k \rangle \in$ f(has-name), and this n is a closed term of the language.

Finally, there is a special quotation operator q. This operator is not a function letter - it is a real extension of first-order syntax. If $v_1 ... v_n$ are variables, $k_1 ... k_n$ are constants in K, and p is any wff of our language, then [q [$v_1 ... v_n$] [$k_1 ... k_n$] p] is a term. It denotes a wff formed by replacing the variables $v_1 ... v_n$ in p with vivid designators for the entities they denote. The constants $k_1 ... k_n$ serve to indicate what kinds of vivid designators are intended.

Suppose John knows what Mary's phone number is. This means that he knows that Mary's phone number is *n*, where *n* is an arabic numeral. We can indicate this in our notation by writing

[∃ n [number n] [know John [q [n] [arabic] [has-phone-number Mary n]]]]

The symbol "know" is an ordinary predicate letter, not a special operator. Its second argument is a term that denotes the sentence John knows. Suppose Mary's phone number is 5766, that is n = 5766. Then the sentence John knows is formed by replacing the variable n in the wff

[has-phone-number mary n]

with a vivid designator of kind arabic - that is, the arabic numeral "5766". The result looks like this:

[has-phone-number Mary 5766]

This is the sentence that John knows.

The denotation of [q [v_1...v_n] [k_1...k_n] r] is a wff formed by substituting closed terms for free occurrences of v_1...v_n in r. If v_1...v_n includes all the free variables of r, the denotation will be a sentence, not a wff with free variables. Propositions in our theory are always sentences of thought language. Therefore if the term [q [v_1...v_n] [k_1...k_n] r]

occurs in the translation of a sentence, $v_1...v_n$ should include all the free variables of r. The fragment will ensure that this is always true.

Let E be an environment such that E(v) is a person p. What is the denotation in E of the term [q [v arabic] [wise v]]? It should be formed by replacing v in [wise v] with a vivid designator for p of kind arabic - but there is no such designator. Such expressions may not produce a practical problem for programs. It is common knowledge that people do not have vivid designators of kind arabic, so is easy for a program to recognize that if v denotes a man then [q [v arabic] [wise v]] should not occur in the translation of any sentence. Nevertheless expressions like these are well-formed, and the semantics must do *something* with them. The easiest thing is to choose an arbitrary term to serve as a dummy name. Therefore choose a term B. If $d \in D$ and k is the denotation of a constant in K, define n(d,k) as follows. If there is a t such that [has-name d t k], then t is unique and n(d,k) = t. Otherwise n(d,k) = B. Thus B is a dummy name that allows us to assign a denotation to [q [v] [k] r] even when v denotes an entity that has no vivid designator of kind k.

We write $s[t_1...t_n/v_1...v_n]$ for the result of simultaneously substituting $t_1...t_n$ for $v_1...v_n$ in s. Then $den([q\ [v_1\ ...\ v_n]\ [k_1\ ...\ k_n]\ r],e,M)$ is $r[n(den(v_1,e,M),den(k_1,e,M))...n(den(v_n,e,M),den(k_n,e,M))/\ v_1...v_n]$. The free variables of $[q\ [v_1...v_n]\ [k_1...k_n]$ are $v_1...v_n$. Obviously the denotation of $[q\ [v_1...v_n]\ [k_1...k_n]$ depends only on the denotations of its free variables.

This definition of the q operator of course implies that substitution of equals will fail inside the scope of q. Suppose

clark-kent = superman

[believe Lois [q [] [] [born-on superman krypton]]]

The second sentence says that Lois believes the sentence [born-on superman krypton] (in this case, the variable list is empty). These sentences do not imply

[believe Lois [q [] [] [born-on clark-kent krypton]]]

This example raises the question of proof methods for our logic. There is no proof method at present, though it may be possible to construct one by generalizing the ideas of Konolige (1986). Fortunately we do not need a general proof method in order to do semantics. We only need to establish that certain patterns of argument are valid or invalid.

3.4 The Fragment

3.4.1 Quantification

The fragment is a definite clause grammar (Pereira and Warren, 1980). DCG allows calls to Prolog, and Prolog allows various extra-logical operations. However, the fragment uses only pure Prolog - that is, definite clauses without extra-logical operations. Pereira and Warren write

s(S0,S) :- np(P,N,S0,S1) vp(P,N,S1,S)

to indicate that for all P and N, a sentence extends from S0 to S if a noun phrase with person P and number N extends from S0 to S1 and a verb phrase with person P and number N extends from S1 to S. In the language of context free grammars, a sentence derives a NP with person P and number N followed by a VP with person P and number N. The notation

$$s \Rightarrow np(P,N)\ vp(P,N)$$

is an abbreviation for the definite clause

$$s(S0,S) :- np(P,N,S0,S1)\ vp(P,N,S1,S)$$

The fragment uses Cooper's device of quantifier storage (Cooper 1982). The major categories S,NP and VP each have two semantic features: the translation, which is a wff or term, and the quantifier store, which is a set of quantifiers that bind the variables in the translation. For a NP the translation is a variable v, and the quantifier store consists of a single quantifier which binds v. For example,

$$np([\forall\ x\ [man\ x]],x) \Rightarrow [every\ man]$$

The quantifier for a NP may be a conjunctive or disjunctive quantifier, containing embedded quantifiers. In this case the embedded quantifiers must all bind the same variable. Thus we have

$$np([[\forall\ x\ [man\ x]] \wedge [\forall\ x\ [woman\ x]]],x) \Rightarrow [every\ man\ and\ every\ woman]$$

but not

$$np([[\forall\ x\ [man\ x]] \wedge [\forall\ y\ [woman\ y]]],x) \Rightarrow [every\ man\ and\ every\ woman]$$

This translation is impossible because the two embedded quantifiers bind different variables.

For a VP the translation is a wff, and the quantifiers in the store bind all the free variables of the wff except for one: the variable that is the translation of the subject. This variable is supplied as an extra semantic argument of the VP - let us say the first argument. Thus we have

$$vp(y,[[\forall\ x\ [woman\ x]]],[loves\ y\ x]) \Rightarrow [loves\ every\ woman]$$

For an S the translation is a wff and the quantifiers in the store bind all free variables of the wff. Thus we have

$$S([[\exists\ y\ [man\ y]],[\forall\ x\ [woman\ x]]],[loves\ y\ x]) \Rightarrow [some\ man\ loves\ every\ woman]$$

The rule that creates sentences can take quantifiers out of the store and place them at the front of the translation of the sentence. This process of applying quantifiers is non-deterministic, and this non-determinism will create the desired scope ambiguities. To make this more explicit, we present the Prolog predicate that takes a list of quantifiers and applies them to a wff. apply(L,Wff1,Wff2) means that applying the quantifiers in list L to Wff1 can produce Wff2. It is defined by the following Horn clauses:

```
apply([],Wff1,Wff1) :-

apply(L,Wff1,[Q Wff2]) :- pick(Q,L,M) apply(M,Wff1,Wff2)
```

pick(X,L,M) means that X is an element of list L and M is the list remaining after X is removed from L. It is easily defined:

```
pick(X,[X | L],L) :-
pick(Y,[X | L],[X | M]) :-  pick(Y,L,M)
```

If we are interpreting the sentence "Some man loves every woman", we will find that there are two solutions to the goal

```
:- apply([[∃ x [man x]],[∀ y [woman y]]],[love x y],Wff)
```

One is

```
Wff =  [[∃ x [man x]] [[∀ y [woman y]] [love x y]]]
```

The other is

```
Wff = [[∀ y [woman y]] [[∃ x [man x]] [love x y]]]
```

This mechanism certainly allows us to get wide-scope readings and create ambiguities. The danger is that it may create too many ambiguities. For example, the sentence "Every man loves the woman" has only one reading, with "the" outscoping "every". The predicates above would allow another reading, in which "every" has the wider scope. This occurs because the predicate "pick" can remove the quantifier ∀ from the quantifier store and apply it before applying the quantifier ∃!. We could prevent this by re-defining the predicate "pick", so that it always applies the quantifier ∃! before other quantifiers. People prefer readings in which the order of scoping is the same as the order of quantifiers in the original sentence. In the fragment quantifiers always appear in the store in the order in which they appear in the sentence, so this preference is easy to represent. Quantifier scope is a difficult problem, and this paper will not offer a solution - we merely indicate where in our system the solution could fit.

Pereira and Shieber (1987) also present a DCG using quantifier storage. The quantifiers in their fragment are properties of predicates, as in Montague, rather than complex first-order quantifiers as in the present fragment. Their fragment builds an intermediate representation called a quantifier tree, which sums up a set of possible readings, rather than generating the various readings directly.

A key problem in translating English to logic is making sure that different quantifiers bind different variables. The translation of "Some man loves every woman" can be

[[∃ x [man x]] [[∀ y [woman y]] [loves x y]]]

but it cannot be

[[∃ x [man x]] [[∀ x [woman x]] [loves x x]]]

Pereira and Warren (1980) solved this problem by using Prolog variables to represent variables of the object language. Thus their translation for "Some man loves every woman" is

exists(Y) : (man(Y) & all(X) : (woman(X) => loves(Y,X)))

where X and Y are Prolog variables. This works, but it violates the declarative semantics of Prolog. According to that semantics every variable is universally quantified. Thus suppose $S(Wff_1,P1,P2)$ means that the text from point P1 to point P2 is a sentence whose translation is Wff_1. If Prolog proves the clause

S(exists(Y) : (man(Y) & all(X) : (woman(X) => loves(Y,X))),P1,P2) :-

this means that for all values of X and Y the expression

exists(Y) : (man(Y) & all(X) : (woman(X) => loves(Y,X)))

is a possible translation for the string from P1 to P2. This means that if v is a variable of the object language, then

exists(v) : (man(v) & all(v) : (woman(v) => loves(v,v)))

is a possible translation - which is clearly false. Thus according to the declarative interpretation, Pereira and Warren's fragment does not express the requirement that different quantifiers bind different variables.

This does not mean that the fragment is wrong - in fact it is provably correct, according to the procedural interpretation that Pereira and Warren intended. However, as Pereira and Warren explained, the declarative semantics has a large advantage over the procedural one. It allows us to prove partial correctness of a Prolog program without using procedural concepts - by understanding the program as a set of statements about the problem domain and verifying those statements. To prove the correctness of Pereira and Warren's fragment one must understand the implementation of Prolog - in particular, the way that variables are renamed each time a clause is used.

In fact it is easy to ensure that different quantifiers bind different variables without using a procedural interpretation. The variables in our translations will be chosen from the sequence $v_0, v_1, v_2 \ldots$, etc. Let the predicates S and VP each take an extra argument N, which represents a lower limit on the variables bound in the translation or the quantifier store. That is, if a quantifier in the translation or the store binds v_M then $M \geq N$. We will use this extra argument to keep track of the variables that have already been bound and make sure they are not re-bound.

To formalize this, let the Prolog term var(N) denote v_N. We represent the integers using the constant 0 and the function letter "s" for "successor". Thus s(s(0)) denotes the number 2, and var(s(s(0))) represents the object language variable v_2. As an abbreviation we write v_2 instead of var(s(s(0))), N+2 instead of s(s(N)), and v_{N+2} instead of var(s(s(N))). Then we have

$S([[\exists\ v_N\ [man\ v_N]],[\forall\ v_{N+1}\ [woman\ v_{N+1}]]],[loves\ v_N\ v_{N+1}],N) \Rightarrow$ [some man loves every woman]

The last argument of the predicate S is the lower limit on the variables bound in the translation or the quantifier store. The subject translation is v_N and the object translation is v_{N+1}. These are distinct variables for all values of N. Thus the clause specifies that the quantifiers bind distinct variables without specifying any particular variables.

Here is an example of a rule that uses this mechanism:

$$S(Q_1,Wff_1,N) \rightarrow NP(Q_2,v_N)\ VP(v_N,Q_3,Wff_2,N+1)$$

(This is an abbreviation of the full rule.) The subject translation is v_N. The lower limit on the VP is N+1, so if v_M occurs in the VP translation or the VP store then M > N. Thus no quantifier from the VP can bind the same variable that the subject quantifier binds.

In its general form this technique requires two extra arguments on each major category - a lower limit and an upper limit. In our fragment embedded S's and VP's always appear at the left end of a rule, and this allows us to simplify slightly by eliminating the upper limit.

3.4.2 The Rules

Each of our rules combines syntax and semantics, but for ease of exposition we first present the syntax of our fragment without any semantics.

R1 start → s

R2 s → np vp

R3 np → quant n

R4 quant → [a]

R5 quant → [every]

R6 quant → [no]

R7 quant → [the]

R8 np → [john], np → [mary], etc.

R9 np → np [and] np

R10 np → np [or] np

R11 vp → v(trans) np

R12 vp → v(takes-s) s

R13 vp → v(intensional) np

R14 n → [italian], n → [pizza], etc.

R15 v(trans) → [eat], etc.

R16 v(take-S) → [believe]

R17 v(intensional) → [want], etc.

This syntax will derive sentences like "Every man loves some woman", "John believes that the cat likes the fish", and so on. The rules R8 and R14-R17 are actually sets of similar rules introducing terminal symbols. Notice that the transitive verb "want" is introduced by a special rule, although its syntax is apparently the same as the syntax of ordinary transitive verbs. This is done because the semantics of "want" is quite different from the semantics of ordinary transitive verbs. Montague represented this difference with a meaning postulate, which saved him from making two copies of the syntactic rule for transitive verbs. We could avoid the duplication by a slight extension of our notation.

We now extend our rules to include semantics, beginning with np's.

R14a n(man) → [man]

R14b n(woman) → [woman]

The symbol "man" is a predicate of one argument. The rules for other common nouns are similar. We have used the same symbol for an English word and a predicate letter in the thought language, which strictly speaking is illegal, but context should make the meaning clear in each case.

The semantic representations of English quantifiers are logical quantifiers.

R4 quant(∃) → [a]

R5 quant(∀) → [every]

R6 quant(~∃) → [no]

R7 quant(∃!) → [the]

The main rules for np's is:

R3 np([Q V [P V]],V) → quant(Q) n(P)

The function letter "np" takes two semantic arguments: a quantifier and a variable. The quantifier binds the variable.

Rules R3, R5, and R14 imply

$np([\forall\ V\ [man\ V]],V) \stackrel{*}{\Rightarrow}$ [every man]

Since V is a free Prolog variable, any variable of thought language can be the translation of the NP.

The rule for conjunctive np's is as follows:

R9 $np([Q1 \wedge Q2],V) \rightarrow np(Q1,V)$ [and] $np(Q2,V)$

The complex quantifier [Q1 $\wedge$ Q2] must bind a single variable, so the two embedded quantifiers must bind the same variable. To ensure this, we set the translations of the embedded NP's equal to the translation of the whole NP. By R3, R4, and R14

$np([\exists\ V\ [man\ V]],V) \stackrel{*}{\Rightarrow}$ [a man]
$np([\exists\ V\ [woman\ V]],V) \stackrel{*}{\Rightarrow}$ [a woman]

Then by R9

$np([[\exists\ V\ [man\ V]] \wedge [\exists\ V\ [woman\ V]]],V) \stackrel{*}{\Rightarrow}$ [a man and a woman]

The rule for disjunctive np's is very similar:

R10 $np([Q1 \vee Q2],V) \rightarrow np(Q1,V)$ [or] $np(Q2,V)$

We now turn to np's without explicit quantifiers.

R8 np([[∃! V [name V john]]],V) → [john]

"john" is a logical constant denoting a name, and [name V john] means that V is an object whose name is John. Thus we accept the common view that "John" means the same as "the person called John". We turn now to vp's and sentences without propositional attitude verbs. Ordinary transitive verbs translate to predicates of two arguments:

R13 v(trans,like) → [likes]

The function letter "vp" has three semantic arguments: a variable V, a quantifier list, and a wff W. The variable V represents the logical subject of the verb phrase. The quantifier list will bind all the free variables in wff W, except for V. The rule for VP's with ordinary transitive verbs is

R11 vp(V,[Q],[P V v_N],N) → v(trans, P) np(Q,v_N)

We have

v(trans,love) ⇒ [loves]
np([∃ V [woman V]],V) ⇒ [a woman]

and R11 gives

vp(V,[[∃ v_N [woman v_N]]],[love V v_N],N) ⇒ [loves a woman]

The rule for sentences is the most complex so far. It collects the quantifiers from the subject and verb phrase into a single list, and splits that list into two sub-lists. The first sub-list is applied to the VP translation to form the sentence translation. The second sub-list becomes the quantifier store of the sentence. If the sentence is not embedded, its quantifier store will be empty. If the sentence is the object of a verb like "believe", its quantifier store contains the quantifers for those np's that receive de re readings. The process of splitting the list of quantifiers is non-deterministic, and this non-determinism gives rise to de re/de dicto ambiguities. The predicate split(L1,L2,L3) means that the list L1 can be divided into sub-lists L2 and L3. The following clauses define this predicate:

```
split([],[],[]) :-
split([X|L],[X|M1],M2) :- split(L,M1,M2)
split([X|L],M1,[X|M2]) :- split(L,M1,M2)
```

Here is the full version of the sentence rule:

R2 S(L3,Wff1,N) → NP(Q,v_N) VP(v_N,L2,Wff2,N+1)
{ split([Q | L2],L1,L3) apply(L1,Wff2,Wff1) }

The subject translation is V_N, while the lower limit for variables in the VP is N+1. The quantifier Q binds the variable v_N, while L2 binds all the variables in Wff2 except v_N, so [Q | L2] binds all the variables in Wff2. This list is split into parts L1 and L3. The quantifiers in L1 are applied to Wff2 to form the translation Wff1 of the sentence, while the quantifier list L3 becomes the quantifier store of the sentence. Since the quantifiers in L1 and L3 together bind all the free variables in Wff2, the quantifiers in L3 bind the free variables that remain after applying the quantifiers in L1.

We illustrate this rule on the sentence "Every man loves a woman". We have

np([∀ V [man V]],V) ⇒ [every man]

vp(V,[[∃ v_M [woman v_M]]],[love V v_M],M) ⇒ [loves a woman]

Unifying with the right side of R2 gives [Q | L2] = [[∀ v_N [man v_N]],[∃ v_{N+1} [woman v_{N+1}]]]. Suppose the predicate "split" returns L1 = [Q | L2], L3 = []. The predicate "apply" can apply the two quantifiers in two different orders, giving either

Wff1 = [[∀ v_N [man v_N]] [[∃ v_{N+1} [woman v_{N+1}]] [loves v_N v_{N+1}]]]

or

Wff1 = [[∃ v_{N+1} [woman v_{N+1}]] [[∀ v_N [man v_N]] [loves v_N v_{N+1}]]]

Therefore R2 gives

S([],[[∀ v_N [man v_N]] [[∃ v_{N+1} [woman v_{N+1}]] [loves v_N v_{N+1}]]]],N)
⇒ [every man loves a woman]

and also

S([],[[∃ v_{N+1} [woman v_{N+1}]] [[∀ v_N [man v_N]] [loves v_N v_{N+1}]]],N)
⇒ [every man loves a woman]

If a sentence forms an utterance, then the quantifier store must be empty. Therefore the full version of rule R1 is

R1 start(Wff) → S([],Wff,0)

We have already shown that the quantifier store of an S binds all the free variables in the translation. If the quantifier store is empty, there are no free variables in the translation. Then the translation of the start symbol is a sentence, which is just the translation we expect for a full utterance. The lower limit on bound variables in the sentence is 0. Combining this rule with the derivation given above we have

start([[$\forall$ v_0 [man v_0]] [[$\exists$ v_1 [woman v_1]] [loves v_0 v_1]]])

$\stackrel{*}{\Rightarrow}$ [every man loves a woman]

This approach to quantifiers requires less formal machinery than Montague's or even Cooper's. There are no functions with complex higher types, only a modest extension of first-order logic. There is no need to build a complex translation and then simplify - this fragment builds a simple translation in the first place. As a result, much of the structure of the original syntax tree is preserved in the translation. Each NP corresponds to a quantifier, and if NP_1 is embedded in NP_2 then the translation of NP_1 is embedded in the translation of NP_2. The quantifiers are moved from their original places in the tree, but the variables bound by the quantifiers serve to mark the original places. Thus in the sentence "Every man loves a woman", the NP "every man" is the subject of the verb "loves" and the NP "a woman" is its object. In the translations given for that sentence the quantifier [$\forall$ v_N] [man v_N]]] binds the variable v_N, which is the first argument of the predicate "loves".

The quantifier [∃ v_{N+1} [woman v_{N+1}]] binds the variable v_{N+1} which is the second argument of the predicate "loves".

These remarks are not theoretical criticisms of Montague or Cooper. The theory of functions of higher type is quite clear. The simplification has no theoretical significance in Montague's or Cooper's work - it is a mere convenience. On the practical side, however, there is a real problem. Montague grammar is very hard to do, and its warmest admirers admit this. As a practical matter it is easier to work with expressions of first-order logic than with functions of higher type, and easier to build a simple translation than to build a complex one and simplify it. Artificial intelligence workers are committed to building large fragments, and for them such practical considerations are important.

We turn now to the treatment of propositional attitudes, beginning with verbs like "believe" and "know" that take sentences as complements. The translations of these verbs are predicates of two arguments:

R16 v(takes-s,believe) → [believe]

Before giving the complete version of R12, we introduce some auxiliary predicates. The function letter "S" has two semantic arguments: a wff, and a list of quantifiers that bind the free variables of that wff. The q operator requires three arguments: a wff, a list of the free variables in that wff, and a list of constants from the set K of special constants. In order to obtain suitable arguments for the q operator we must take the quantifiers from the S, find the variables that they bind, and construct a list containing a constant from K for each variable. q-args(L1,L2,L3) means that L1 is a list of quantifiers, L2 a list of variables bound by those quantifiers, and L3 a list of constants from K of the same length as L2.

This predicate will take the semantic arguments of "S" and construct suitable arguments for the q operator. Its definition is

```
q-args([],[],[]) :-
q-args([[Q V P] | L],[V | M1],[K1 | M2]) :- in-k(K1) q-args(L,M1,M2)
```

in-k(K1) means that K1 is a member of the set K of special constants. It is defined by axioms like:

```
in-k(arabic) :-
in-k(selfname) :-
```

Using these auxiliary predicates we can state the full version of R12.

```
R12 VP(V,L1,[P V [q L2 L3 Wff1]],N)
        → V(takes-S, P) S(L1,Wff1,N)
    { q-args(L1,L2,L3) }
```

"q-args" will set L2 to a list of the free variables in Wff1, and L3 to a list of constants from K, with L3 the same length as L2. Therefore the expression [q L2 L3 Wff1] is well-formed and denotes a sentence.

Consider the VP "believes an Italian discovered America". We will simplify the example by assuming that the translation of "America" is a constant, rather than a variable bound by the quantifier ∃!. We have

S([],[[∃ v_N [italian v_N]] [discover v_N america]],N)

⇒ [an Italian discovered America]

and also

S([[∃ v_N [italian v_N]]],[discover v_N America],N)

⇒ [an Italian discovered America]

In the first derivation, the quantifier [∃ v_N [italian v_N]] has been applied to the wff. In the second derivation it is still in the quantifier store. Both derivations are possible because the rule for sentences can either apply the quantifiers to the wff, or put them in the store.

Suppose we derive the object of "believe" by the first derivation. Applying rule R12 we find that L1 is the empty list, and therefore L2 and L3 are empty. Therefore we have

VP(V,[],[believe V [q [] [] [[∃ v_N [italian v_N]] [discover v_N america]]]],N)

⇒ [believes an Italian discovered America]

In this case the quantifier is inside the scope of the belief operator, and the NP "an Italian" has a de dicto interpretation.

Now suppose we derive the object of "believe" by the second derivation. Then L1 = [[∃ v_N [italian v_N]], L2 = [v_N], and L3 has the form [k_0], where k_0 is an element of the set K of special constants. We get

VP(V,[[∃ v_N [italian v_N]]],[believe V [q [v_N] [k_0] [discover v_N america]]])

⇒ [believes an Italian discovered America]

Here the quantifier is outside the scope of the belief operator, but it binds a variable inside the scope. According to our theory, this means that the agent must have a vivid designator for the person who he believes discovered America. One obvious choice is [ι x [name x columbus]] (assuming that we have introduced the ι operator into our logical language). Then the agent's belief would be

[discover [ι x [name x columbus]] America]

We now consider intensional verbs like "want". In our theory a propositional attitude is a relation between agents and sentences of thought language. Therefore in our treatment, the verb "want" must build a thought-language sentence from the semantic features of its object. We can see how this is to be done by noting that "John wants a Ferrari" is roughly synonymous with "John wishes that he had a Ferrari". Our formalism cannot treat this sentence as it stands, since we have no treatment of pronouns. Still we can write an expression of thought language that represents its meaning. If the existential quantifier is inside the scope of the verb "wish", the representation is

[[∃! x [name x john]]
[wish x [q [x] [k_1] [∃ y [ferrari y] [have x y]]]]]

where k_1 is a constant denoting a kind of vivid designator. This says that John wishes that a certain sentence be true, and that sentence has the form

[[∃ y [ferrari y]] [have t_1 y]]

where t_1 is a vivid designator for John himself. This raises the question "Which vivid designator?", and our answer naturally is that it must be John's selfname - the name he standardly uses to refer to himself in his own beliefs and goals.

Given the tools we have developed so far, these ideas can easily be formalized. The representation of an intensional verb like "want" consists of two predicates. The first predicate represents a propositional attitude, the second one is the main predicate of the sentence which is the object of that attitude. For "want" we have

R17a V(intensional,wish,have) → [want]

This means that "John wants X" is roughly synonymous with "John wishes he had X" (the predicate letters "wish" and "have" need not be exactly synonymous with the English words). For "seek" we have

R17b V(intensional,attempt,find) → [seek]

This means that "John seeks X" is roughly synonymous with "John attempts to find X".

Here is the full version of R13.

R13 VP(V,L2,[P1 V [q [V | L3] [selfname | L4] Wff1]],N)
→ V(intensional,P1,P2) NP(Q,v_N)

{ split([Q],L1,L2)
apply(L1,[P2 V v_N],Wff1)
q-args(L2,L3,L4) }

This rule is complex, but it is easier to understand if we see that it combines the actions of the rule R2 for sentences and the rule R12 for verbs like "believe". The rule first takes the list [Q] containing the quantifier from the object NP and splits it into two parts L1 and L2 (since [Q] has only one element, either L1 = [Q] and L2 = [] or L2 = [Q] and L1 = []). The quantifiers in L1 are applied to the wff [P2 V V_N] to form Wff1, while the quantifiers in L2 go into the quantifier store. This is much like the rule for sentences.

The free variables of [P2 V V_N] are the variables bound by L1 and L2, plus the subject translation V. After the quantifiers in L1 are applied, their variables are no longer free. Therefore, the free variables of Wff1 are the variables bound by L2, plus V. The rule now uses "q-args" to construct a list L3 of variables and a list L4 of special constants from K. This is like the action of rule R12 for the verb "believe", but there is a difference. L3 and L4 are not suitable arguments for the q operator, because L3 does not include all the free variables of Wff1. It leaves out V. Therefore the rule inserts [V | L3] and [selfname | L4] as the arguments of the q operator. This indicates that the selfname denotes the agent in the proposition.

This rule gives two readings for the VP "wants a Ferrari". We have

NP([∃ V [ferrari V]],V) $\stackrel{*}{\Rightarrow}$ [a Ferrari]

v(intensional,wish,have) → [want]

Unifying with the right side of R13 gives [Q] = [[∃ v_N [ferrari v_N]]]. Suppose that the predicate "split" sets L1 = [[∃ v_N [ferrari v_N]]] and L2 = []. Then the predicate "apply" sets Wff1 = [[∃ v_N [ferrari v_N]] [have V v_N]]. Since L2 = [], the predicate "q-args" sets L3 = [] and L4 = [], and we get

VP(V,[],[wish V [q [V] [selfname] [[∃ v_N [ferrari v_N]] [have V v_N]]]],N)

⇒ [wants a Ferrari]

In this reading the quantifier is inside the scope of the propositional attitude. Suppose the selfname of the subject is Me[1]. Then the object of the propositional attitude "wish" is

[[∃ v_N [ferrari v_N]] [have Me_1 v_N]]

We can get the other reading, with the quantifier outside the propositional attitude, as follows. Suppose that the predicate "split" divides [[∃ v_N [ferrari v_N]]] into L1 = [] and L2 = [[∃ v_N [ferrari v_N]]]. Then Wff1 = [have V v_N], and L3 = [v_N]. Therefore L4 has the form [K1], where K1 is a constant from K denoting a kind of vivid designator. It is not clear what kind of vivid designator would be appropriate for a car, but that problem is outside the scope of this paper, so assume that $k_0 \in K$ and let K = k_0. With L4 = [k_0], we get

VP(V,[[∃ v_N [ferrari v_N]]],[wish V [q [V v_N] [selfname k_0] [have V v_N]]],N)

⇒ [wants a Ferrari]

Let us combine this VP with the subject "John" to form a sentence. We have

NP([∃! V [name V john]],V) ⇒ [John]

Applying the sentence rule to this NP and VP gives

Wff2 = [wish v_N [q [v_N v_{N+1}] [selfname k_0] [have v_N v_{N+1}]]]

The sentence rule will also apply the quantifiers to this wff. The order of application does not matter in this case - both possible answers have the same truth conditions. One answer is

[[∃! v_N [name v_N John]]
[[∃ v_{N+1} [ferrari v_{N+1}]]
[wish v_N [q [v_N v_{N+1}] [selfname k_0] [have v_N v_{N+1}]]]]]

This says that there is a particular Ferrari that John wants. Suppose that the vivid designator that John uses to describe that Ferrari is f_1, and John's selfname is Me_1. Then the object of "wish" is the sentence

[have Me_1 f_1]

which contains no quantifiers.

As a final example consider "John wants a Ferrari or a Porsche". This sentence has a reading which means that he wants any Ferrari or Porsche. The following sentence of thought language expresses the meaning of this reading:

[[∃! v_0 [name v_0 John]]
[wish v_0 [q [v_0] [selfname]
[[[∃ v_1 [ferrari v_1]] ∨ [∃ v_1 [porsche v_1]]] [have v_0 v_1]]]]]

The fragment generates this translation, as the reader can check.

3.5 Conclusion

We have shown that a sentential semantics for propositional attitudes can account for quantifiers that stand outside the scope of a propositional attitude and bind a variable inside that scope. Our fragment includes related phenomena: failure of substitution in the scope of propositional attitudes, and the scope ambiguity in examples like "John wants a Ferrari". Our treatment is compositional in the sense that the semantic features of a phrase are constrained by the semantic features of its immediate constituents, not by global properties of a parse tree. On the other hand, our notion of "semantic feature" is quite broad - a variable denoting the logical subject is one of the semantic features of the verb phrase. Perhaps this violates compositionality.

It seems pointless to argue about whether our treatment is *really* compositional. Instead we will argue that our formalism has one very important property in common with Montague's and Cooper's. There is a systematic connection between the syntax of a phrase and its semantics, and this connection allows us to check that each rule gets input it can accept, and that the final result makes sense. For example, we showed that the quantifier store of a sentence always contains the quantifiers needed to bind all the free variables in the

translation. This is true because the quantifier from the subject binds the variable that represents the subject, while the quantifiers from the vp bind all the other variables.

In other ways our treatment is very different from the possible worlds formalisms. We translate to a first-order logic augmented with a quotation operator, rather than an intensional logic. This language is at the center of our theory, because we claim that beliefs and goals are sentences of the language. Montague regarded his intensional logic as a mere convenience - his goal was to assign denotations in terms of possible worlds. Cooper followed this line the whole way, abolishing the intensional logic and mapping sentences directly to their denotations.

Montague and Cooper used possible worlds not just for their original purpose, to describe possibility and necessity, but also to treat propositional attitudes and even adverbs. We claim that this is a mistake. Possible worlds are an excellent tool for analyzing possibility, necessity, and counterfactuals. They are not a semantic panacea. In particular, they are irrelevant to the study of propositional attitudes.

As Moore said, a fragment based on a sentential theory must treat quantification into the scope of an attitude quite differently from ordinary quantification. In fact we interpret "quantifying in" as the substitution of a vivid designator for a variable. Moore is of course right to argue that other things being equal, we prefer a uniform treatment of quantification. However, other things are not equal. It seems clear that what counts as a vivid designator depends on context, and our theory allows for this by admitting extra arguments to the q operator - arguments which specify which kind of vivid designator is intended. The fragments based on possible worlds do not allow the meaning of "quantifying in" to depend on context.

The extra arguments to the q operator lead to a proliferation of readings for sentences with quantifying in. However, one can form a compact representation for the whole set of readings by putting variables in the second argument of the q operator. For example, one can write

[[∃! v_0 [name v_0 John]]
[[∃ v_1 [ferrari v_1]]
[wish v_0 [q [v_0 v_1] [selfname K1] [have v_0 v_1]]]]]

to represent the set of all translations for the de re reading of "John wants a Ferrari". The free variable K1 ranges over kinds of vivid designators, and the program may be able to choose a value for K1 from knowledge of the context without substituting each possible value into the expression. This shows that the large number of translations generated by the fragment does not automatically lead to computational problems.

This fragment has established that the sentential theory of propositional attitudes can support a semantics for a fragment of English. In future work we hope to add "knowing what" constructions and tenses. We also hope to develop proof methods for our logic, leading finally to a question-answering program that can reason about propositional attitudes.

References

1. Cooper, Robin. Quantification and Syntactic Theory. D. Reidel Publishing Company, Dordrecht.

2. Churchland, Paul M. (1979). Scientific Realism and the Plasticity of Mind. Cambridge University Press, Cambridge.

3. Haas, Andrew. A Syntactic Theory of Belief and Action. Artificial Intelligence 28 (1986), pp. 245-292.

4. Kaplan, D. Quantifying In. in L. Linsky (Ed.), Reference and Modality (University Press, London, 1971) 112-144.

5. Konolige, K. A Deduction Model of Belief. Morgan Kaufmann Publishers, Los Altos, California, 1986.

6. Moore, R. C.. Propositional Attitudes and Russellian Propositions. Report No. CSLI-88-119. Center for the Study of Language and Information, Stanford, California, 1988.

7. Levesque, H. J. A Logic of Knowledge and Active Belief. In Proceedings of the AAAI, University of Texas at Austin, 1984.

8. Montague, R. English as a Formal Language. in Linguaggi nella Societa nella Technica, B. Visentini et. al., eds., pp. 189-224 (Edizioni di Comunita, Milan, Italy, 1970).

9. Pereira, F. C. N., and D. H. D. Warren. Definite Clause Grammars for Natural Language Analysis. Artificial Intelligence 13 (1980), pp. 231-278.

10. Perlis, D. Languages with Self-Reference I: Foundations. Artifical Intelligence, 25(1985), pp. 301-322.

4. Aiding Design with Constructive Plan Recognition

Bradley A. Goodman, BBN Systems and Technologies Corporation

Diane J. Litman, AT&T Bell Laboratories Inc.

4.1 Introduction

We are working towards an intelligent interface to a complex application system that enhances the abilities of a human. The enhancement comes from providing the system with a better notion of what the user is doing (his "plan") so that the system can unburden the user and perform some of the operations itself. These improvements are evidenced through graphical and menu-driven interactions between the system and the user. The system will serve as a cooperative partner with the human in the performance of the task and during any error recovery. Computer aided design systems provide a perfect forum for applying such an intelligent interface since the human designer is creating his design on the fly. We have chosen chemical process design as our domain for demonstrating such an intelligent interface.

Success in such a system involves efficiently designing an application domain process, such as a chemical plant, through *fully integrated human-machine interactions* with a computer aided design system, and not simply passing the human's design from one isolated subcomponent (e.g., a graphics drawing program) to another (e.g., a simulator).

Success also requires adapting the system to the particular user under a variety of application domain conditions. Advances in interface technology will improve overall system performance and result in more efficient human-machine interactions; advances in system adaptation will enhance the accuracy of the system and increase the user's performance. Success, however, will occur not simply because of the graphics interface technology but due to the added strength derived from plan recognition as part of a new level of tools provided to the user. We are providing a new way of applying and using this technology for design.

Design is a creative task to formulate an arrangement of elements to define some process. Today's commercial design tools come in two categories: computer aided design drawing tools and computer aided design simulators. The drawing tools are graphics programs that allow the designer to draft the design on the display. The computer aided design simulators provide numerical simulations of the process being designed to allow the designer to both evaluate his design and specify it more precisely.

What is missing from the current generation of design systems is an intelligent interface that assists the designer during the generation of the graphics representation of the design. While the numerical simulator helps determine the validity of the design itself, an intelligent graphics interface can assist the designer incrementally as he composes the design. It can advise him, as he proceeds, that what he is doing looks reasonable both locally (e.g., two components are correctly connected) and globally (e.g., the design fits in with other designs already known by the system).

With plan recognition we will provide another layer of sophistication to the user interface technology to make design systems much easier to use while making them more powerful.

Our plan recognition tools attempt to surmise the purpose of the user's actions and the particular design he is producing. Understanding the user's purpose allows the system to catch mistakes by the user, to produce the desired effect from less than appropriate actions, and to provide shortcuts where appropriate. By attempting to infer and monitor the user's design over time, the system could cut down design time by recognizing and suggesting where the user could use existing subassemblies (found in the system's design library), prevent redundancy, acquire new designs for future sessions, and detect and recover from invalid designs.

In the first section of the paper we describe how to extend current plan recognition technology to provide the required support to an intelligent interface. The second section then outlines how such a plan recognizer can augment a computer aided design system.

4.2 Extending Plan Recognition

In a typical plan-based interface system, a plan recognizer interprets a sequence of actions by constructing an underlying plan structure. This structure is then analyzed and manipulated in order to extend the capabilities of the associated natural language (or other) interface. Unfortunately, current plan recognition systems have little to say about *novel plans*, defined here to be novel sequences of actions achieving a novel set of goals. For example, plan recognizers typically search for *known* goals that are explained by novel action sequences, or "parse" observed actions in terms of *precompiled* action sequences. In other words, plan recognizers typically assume that they have complete and predefined knowledge of a user's plans or goals. As a result, plan recognition just entails fitting an observed user action into an expected (implicitly or explicitly) user plan.

In this section we present an approach to plan recognition that relaxes the assumption that the system's knowledge of the user's plans and goals is complete. By removing this assumption, our plan recognition algorithm cannot be limited to matching into a predefined set of expected plans or goals. It must be more *constructive* in nature and attempt to fill in potential knowledge gaps. It must do this by attempting to distinguish a purposeful plan from a sequence of random actions and objects.

We illustrate in Section 1.3 the utility of *constructive plan recognition* in the context of a plan-based process design system. Design appears to be an excellent forum for applying plan recognition. Recent research projects in AI and design (Tong, 1987a) have expanded design systems from mere bookkeepers to active participants in the design process. We believe that a plan-based system can add a further layer of sophistication to design interfaces, e.g., by recognizing known designs, determining when new designs are reasonable, and providing diagnostic support when they are not. Furthermore, design appears to be especially suited for the recognition and acquisition of novel plans: since the user is typically creating new designs, the system's design library is inherently incomplete.

4.2.1 Taking Plan Recognition Seriously

Plan recognition algorithms use knowledge of likely plans to infer the intent behind particular input actions. The use of plan recognition to enhance both natural language and other interface systems has been widely documented in the artificial intelligence literature (Allen and Perrault 1980, Bruce 1987, Carberry 1985, Cohen 1978, Genesereth 1979, Kautz 1986, Carver et al. 1984, Litman 1987, Pollack 1986a, Sidner 1985, Schmidt et al. 1978, Wilensky 1983). Several strategies have been employed to recognize and track a user's plan (Allen 1979, Sidner 1985, Carberry 1985, Kautz 1986). These plan

recognition strategies use a plan library: a description of typical plans that might occur in a particular domain. On the basis of the library and description of an action, plan recognition algorithms produce (possibly partial) descriptions of the corresponding plan. Allen's algorithm (Allen 1979) uses a heuristic search to choose a preferred plan, while others (such as Sidner's (1985) or Kautz's (1986)) allow the recognition to occur incrementally.

None of these implementations, however, have been embedded inside a complete working application. As a result, crucial issues of robustness, reliability and inherent limitations remain. In particular, most current algorithms make the incorrect assumption that valid and complete plan knowledge is specified and shared by all agents, and they do not consider the fact that users often make mistakes or get sidetracked. Our research attempts to rectify these deficiencies.

First, we wish to remove the assumption that the system's knowledge of the user's plans is complete. This is especially necessary in the context with which we are concerned, computer aided design (CAD), where the user is often creating new plans (i.e., new designs). If we remove this assumption, however, our plan recognition algorithm cannot be limited to matching into a predefined set of expected plans. It must be more *constructive* in nature and attempt to fill in potential knowledge gaps when presented with novel plans.[1] It must do this by attempting to distinguish a purposeful plan from a sequence of random actions and objects. For example, constructive plan recognition should be able to use local knowledge about primitive actions, such as effects and preconditions,[2] to knit actions

[1]Following in spirit the Hypothesize and Revise paradigm employed in Believer by Schmidt et al. (1978) and the plan reconstruction employed in Genesereth's automated consultant (Genesereth 1979).

[2]In the traditional planning representations that form the basis for most plan-based systems, action descriptions are represented as operators on a world model (Fikes and Nilsson 1971, Sacerdoti 1974). Preconditions are conditions that need to hold in the world model before the action operator can be applied. Effects are statements that are asserted into the world model after the action has been successfully executed.

together into a complete plan. It should also be able to use what incomplete knowledge of higher level plans it has to acquire new plans. Unlike previous plan recognition approaches, CPR thus marries (rather than separates) plan generation, plan acquisition, and plan recognition.

Second, plan recognition inherently involves communication between agents, whether it is done linguistically or otherwise. However, agents have imperfect skills and can be sloppy. Thus, communication is often approximate and even errorful, leading to possible miscommunication (Goodman 1986). For example, even in the simplest plan recognition scenario where an agent successively executes steps in a *shared* plan, while another agent (the system) successively tries to reconstruct and track the user's plan based on the user's actions, the user can make mistakes, e.g. by executing the wrong action in a plan. If the system cannot construct any reasonable plan from the user's actions, CPR will use the source of disparity to help detect and repair the miscommunication. Current plan recognition systems, in contrast, would just fail.

Although other implications of multiple agents remain, we will not focus on them at this time. For example, while we relax the assumption that knowledge is complete across agents, we will assume that any remaining shared knowledge is correct and still consistent across agents. For work that explicitly focuses on the relaxation of various knowledge correctness assumptions to repair invalid plans, see Pollack (1986a, 1986b) and Carberry (1985, 1986). Pollack took a major step forward by dropping that assumption and demonstrating how to handle misconceptions based on action disagreements. She provided a new model of plan inference based on the analysis of plans as mental phenomena. Carberry also departs from the previous views. She is concerned with user models and how differences between those of the speaker and hearer can lead to disparities in the

speaker's plan and the one inferred from the speaker's utterance by the hearer. She argues that handling disparate plans requires an enriched context model. She describes in (Carberry 1986) how to drive a negotiation model to make the models of the speaker and hearer agree. We also assume that plans do not require true collaborative behavior amongst agents. Again, we will see that this is a reasonable assumption to make in the context of an intelligent CAD system. For relaxation of this "master-slave" assumption, see the work of Grosz and Sidner (1987).

4.2.2 Constructive Plan Recognition

Existing plan recognition techniques operate in a "syntactic" framework. They perform a process similar to parsing by attempting to fit observed user actions into an expected user plan (as defined by a joint library of plans or goals), not unlike a language that defines all possible sentences (cf. Sidner (1985), Huff and Lesser (1982), Carver et al. (1984)). CPR attempts to extend this parsing analogy by developing a cascaded parsing algorithm (Woods 1980). However, while a traditional cascaded parser uses semantics to verify or eliminate existing syntactic choices, our algorithm uses semantics to *construct* additions to the incomplete syntactic language. That is, if CPR cannot parse an observed action sequence, it attempts to determine whether or not the sequence could be part of the plan language (i.e., is "purposeful") by applying semantic information.[3] Purposeful actions must be able to be seen as being part of an action sequence on the way towards achieving some goal. In the case of novel sequences, however, that goal is not necessarily known.

[3]Allen and Perrault (1980) suggest a similar scheme but they assume a non-null expectation and always recognize a complete plan.

CPR uses *plan generation* and *plan modification* techniques to determine whether novel sequences of actions are potentially "purposeful." For example, CPR incrementally examines (using local backward and forward chaining) the effects and preconditions of actions in an action sequence and the propagation of those effects to determine whether or not they fit together well.[4] However, when effects of earlier actions neither violate nor achieve preconditions of later actions, CPR can say nothing definitive about the causal structure of the plan. Instead, the desired interactions must be viewed as expectations that need to be satisfied before the plan specification is through in order for the action sequence to remain valid. Because CPR uses bottom up techniques to incrementally *verify* the coherence of action sequences, the search explosion involved in earlier bottom up approaches to plan recognition (where search spaces were generated to *propose* action sequences) can be controlled.

Alternatively, failed "plan parses" from an incomplete library, along with techniques of case-based and adaptive reasoning (e.g., Alterman (1986), Broverman and Croft (1987), Hammond (1987), Kass (1986), Kolodner et al. (1985), Koton (1988), and the DARPA Workshop on Case-Based Reasoning (1988)), can also be used to determine if novel actions and goals are reasonable plans. However, the integration of plan modification techniques with plan recognition from first principles is an interesting open question. As will be seen in Section 1.3, we currently leave this control problem in the hands of the user.

[4] If an effect of one action violates a precondition of a later action in the sequence, then a plan generator determines if another action (other than directly undoing the original action) could be hypothesized to restore the desired precondition. If so, the input is considered potentially purposeful.

Finally, if after application of the above techniques an action sequence still cannot be considered purposeful, CPR assumes that an error has occurred. The system interrupts and calls a *plan relaxation* component that uses the information present in the failed recognition attempt to provide diagnostic support to debug the plan. This component follows in spirit the relaxation mechanism developed by Goodman (1986) to deal with extensional reference failures.

4.2.3 An Example

Consider the following simple example to illustrate CPR. Figure 1 illustrates a portion of a plan hierarchy containing plans about Naval operations. The top-most plan is "END." In the lower part of the hierarchy we describe three plans: Protect, Exercise, and Rescue. Decomposition of each plan into their respective action sequence is also shown (the small arrows point to the actions that are to be executed in the sequence from top to bottom). In Figure 2 we provide more detailed specifications of some of the actions composing the plans.

Assume we observe the actions by a ship and its crew shown in Figure 3. Consider first the case where the bracketed step is missing: the ship $ship_1$ moves from location loc_1 to a new location loc_2, patrols around the area $region_1$, tests one of its systems system1, and then returns to its originating location loc_1. Then the action sequence we have is an instance of the Exercise plan since it matches each of its steps.

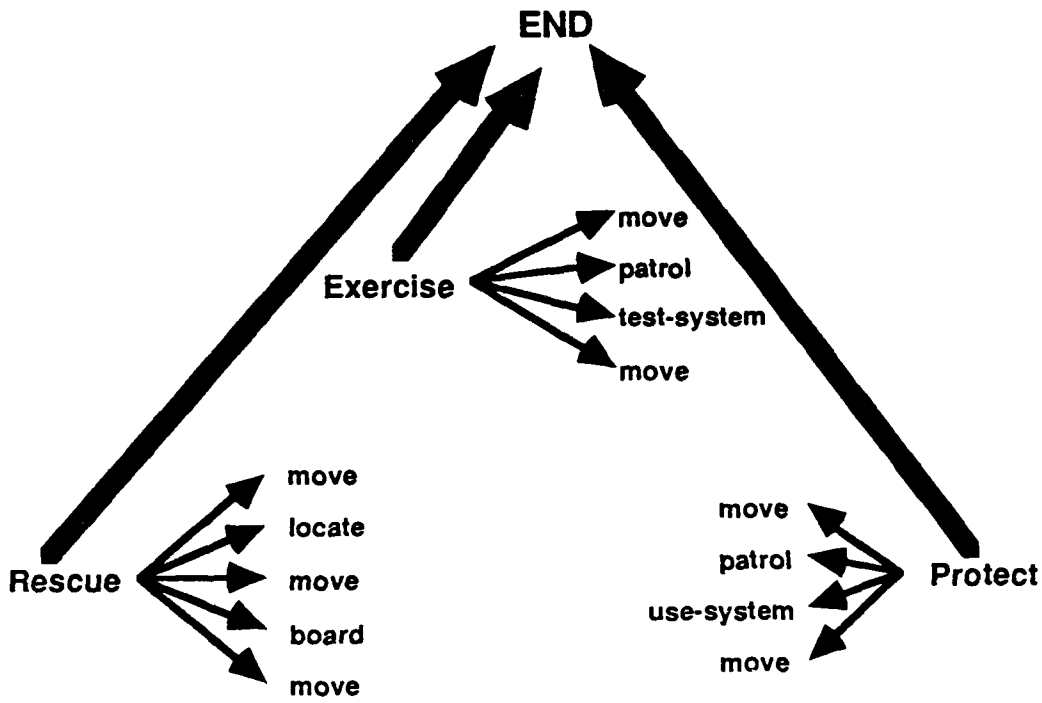


Figure 1: An action and decomposition hierarchy of naval operations

```
Test-System(sh:ship,sy:system,t:time)
  descr:        "Test system sy on ship sh at time t"
  precondition: have(sh:ship,sy:system,t:time)
  effect:       know(c:commander,status(sh:ship,sy:system,t:time))

Move(sh:ship,loc1]:location,loc2:location,t:time)
  descr:        "Move ship sh from location loc1 to
                 location loc2 during time t"
  precondition: at(sh:ship,loc1.location,t1:time)
  effect:       at(sh:ship,loc2:location,t2:time), t2>t1
```

Figure 2: Defining some naval actions

move(ship1:ship,loc1:location,loc2:location,t:time)
patrol(ship1:ship,region1:region,t':time)
test-system(ship1:ship,system1:system,t1:time)
[test-system(ship1:ship,system2,system,t2:time)]
move(ship1:ship,loc2:location,loc1:location,t":time)

Figure 3: A sequence of naval actions

Now suppose the bracketed step, "test system2," is present. The expanded action sequence would not be recognized as any of the the plans defined in Figure 1 under the standard plan recognition algorithms since the action sequence no longer matches *exactly* any plans in the action and decomposition hierarchy.[5] With CPR, however, the plan could be recognized as a reasonable and novel plan since, for example, the effects of the first "test-system," (know[c:commander,status(ship1:ship,system1:system,t:time)]), do not conflict with the preconditions of the second, (have[ship1:ship,system2:system,t:time]). CPR, hence, views the action sequence as *potentially* purposeful. By contrasting the effect of the second "test-system" to the complete "Exercise" plan in the hierarchy, it could even conjecture that an "Exercise" plan has occurred. This conjecture is reasonable because the second "test-system" does not change the effect or negate any precondition of any other action in the "Exercise" plan. In Section 1.3, we describe other ways for CPR to extend the incomplete syntactic knowledge using semantics.

[5]Actually, the Kautz plan recognition algorithm (Kautz 1987) would be able to recognize the plan as two overlaid plans that share steps. Hence it would recognize the sequence as two DIFFERENT exercises being performed.

4.2.4 Implications of CPR

Constructive plan recognition involves problem solving where one induces on the fly semantic generalizations on the plan library hierarchy. Instead of expending a lot of effort ahead of time to encode a very precise and complete hierarchy with lots of abstract entities represented, we form (and thus "learn") the generalizations on the basis of semantics. The burden, thus, is moved from the library to extensive descriptions of individual actions and their effects from which plans in the library are composed. Such actions, however, are much easier to isolate, especially in domains such as computer aided design where the actions are limited to those provided by the software (e.g., a graphics action such as "add icon") and those induced by the application domain itself (e.g., "connect pipe" or "react" in a chemical process domain). This switch in emphasis away from the library specification is a necessary change for computer aided design since designers typically create *new* designs not yet in the library.

The CPR algorithm, thus, entails searching libraries of plans and actions, matching (partially) against entries in those libraries, and chaining through the effects and preconditions of actions. CPR "parses" observations to fit a certain structure and evokes semantics when it fails. Its plan grammar, thus, is incomplete. One possible approach to implementing CPR is to view it as a chart parsing problem. In that model, the constituents being parsed are not words or phrases, but actions and plans.

Parsing algorithms generally have a complexity that is polynomial tractable. True plan parsing schemes, too, are polynomial tractable. They can recognize in polynomial time any underlying plan structure that may exist in the input. They do not attempt, however, to find all possible constituents that could go together. The Kautz (1986) algorithm goes beyond

finding partial constituents and attempts to find *all* the covering sets over the entire input. This methodology allows it to recognize, for example, simultaneously executed plans or plans with shared steps. The increase in coverage, however, leads to possible exponential combinations. With CPR, we are interested in the more tractable case where one gives up a complete covering of the input. For design, however, that makes sense since it would be unrealistic to find such a covering in the first place.

4.3 Aiding Design with Plan Recognition

Numerous AI design tools are being developed. Most of them embrace an expert system framework where the user provides specifications to the design tool and the tool then uses its expert knowledge to develop a design that meets the user's requirements. These include the more knowledge-based approaches that view design as a problem-solving activity (c.f. (Brown86,Mittal86)). Such approaches are making numerous advances in computer aided design but we feel that they are after the very difficult and computationally complex task of *automatic design*. We attempt in this work to view design as a *partnership* between the human designer and the system. That way we can make more progress by simplifying the problem to one whose solution would still be very useful to the human designer. We would also address an area currently neglected by AI researchers in the design task, providing tools for permitting design capture (Tong 1987a). Plan recognition provides the key to these advances.

4.3.1 Taking Design Seriously

The advent of computer aided design emphasizes the need for friendly interfaces to design tools, i.e., interfaces between the designer and the design tools to facilitate such interactions. Natural language interfaces are one possibility (Samad 1986) but standard graphic interfaces make even more sense. Most design domains involve physical objects that have standard graphic symbols that are used by designers on paper to sketch out a proposed design. Most CAD systems require the designer to convert his graphical sketch into a language that represents the essence of the design. For example, one could translate different electronic components and electrical connections between those components into a data structure that described the components and connections. The early CAD systems required the designer to make this translation by hand. More and more CAD systems today, however, are graphics-based and provide their own translation (actually a kind of parsing) to the data structures then used by design tools such as a simulator. The graphics system is more adept to handle the numerous parameters as designs become more and more complex.

Today's commercial design tools tend to come in two categories: CAD drawing tools and CAD simulators. The drawing tools are graphics programs that allow the designer to draft the design on the display. The program can draw and manipulate complex graphical objects to precise scales. The program can also perform bookkeeping for the designer, remembering dimensions, connections between components, and so forth. The CAD simulators provide numerical simulations of the process being designed to allow the designer to both evaluate their design and specify it more precisely (e.g., in a chemical

process, you may be able to specify precisely the reaction and layout components but you might not know how large a storage vessel is necessary until you run the simulator).

What is missing from the current generation of design systems is an intelligent interface that assists the designer during the generation of the graphics representation of the design. While the numerical simulator helps determine the validity of the design itself, an intelligent graphics interface can assist the designer incrementally as he composes the design. It can advise him, as he proceeds, that what he is doing looks reasonable both locally (e.g., two components are correctly connected) and globally (e.g., the design fits in with other designs already known by the system).

4.3.2 Constructive Plan Recognition for Design

We believe that computer aided design is an excellent forum for applying constructive plan recognition. In a typical CAD system (Brown 1986, Mittal 1986, Tong 1987a, Tong 1987b), a user employs a design tool to incrementally assemble portions of a design (i.e a plan) on a graphics display. The tool, in turn, could provide support for the design process, by providing editing facilities, simulations to evaluate a design, record keeping and enforcement of constraints among segments of the design, and access to a design library. Since the user is typically creating *new* designs, the system's design library is inherently *incomplete*. CAD is thus an especially robust domain for CPR. Furthermore, we propose to use CPR to provide another layer of sophistication to the user interface technology. By attempting to infer and monitor the user's design over time, the system could cut down design time by recognizing and suggesting where the user could use existing subassemblies (found in the design library), prevent redundancy, acquire novel designs for future sessions, and detect and recover from miscommunication. For example,

a chemical engineering CAD system could automatically generate, for some of the user's design steps, the most probable continuation of the design by the user and display it dimly on the screen. The user could then either choose the system's suggestion or continue differently. Suppose the user places a Still on the display. A Still has a precondition of a heated mixture on its input. If there are no outputs so far that are heated, the system could then add a "phantom" Heat Exchanger on the Still's input. Similarly, CPR can resolve ambiguities in a design. If the user adds a Still to the output of an already placed Heat Exchanger (a component which can be used in both heating and cooling actions), that requires that the Heat Exchanger "heat" and not "cool" since a precondition of a Still is a heated input.

To achieve such assistance, we need to describe actions and effects at several levels. We must know the kinds of actions the design tool can perform (such as manipulation of graphic symbols) as well as actions that occur in the domain our design tool is assisting. We can provide a means for the user to specify high-level goals that he is attempting to satisfy. Even though his complete plan is not available to the system, it can generate portions of the plan on the basis of the high-level goals and the actions requested of the system by the user.

Our proposed application of CPR to CAD involves a chemical engineering workstation developed to assist in the design of chemical plants. The workstation provides graphic support for putting together the representations of the physical components of the plant (e.g., pipes, vessels, centrifuges, reactors), a process description language for specifying the chemical process to take place, and a simulator for testing out the process specified by the user. CPR can help determine if a partial design is reasonable and provide diagnostic support when it is not. When CPR recognizes that the user is adding a particular

subassembly to the plant design, it can provide a shortcut for the user by allowing him to let the system flesh out the rest of the details of the subassembly. The overall goal is for CPR to help the user as unobtrusively as possible.

The plan-based system that we propose uses the graphics description, general knowledge about what makes a coherent plan (e.g., precondition and effect chains), and a potentially incomplete library of specific designs to build up a functional description of the associated chemical process. In our CAD scenario, a user is constructing a design (plan), and the system is assisting based on the design (plan) library informally defined in Figure 4.

The action hierarchy consists of the two plans REACTION-1 and REACTION-2 and the six primitive actions REACT, TRANSFER, COOL-IN-REFRIG-STORE, COOL-BY-HEAT-EXCHANGER, HEAT-BY-HEAT-EXCHANGER, and DISTILL. In particular, REACTION-1 consists of three steps: the reacting of two substances A and B (resulting in the creation of C), the transferring of C, followed by the cooling of C in a cooling unit. REACTION-2 similarly is composed out of the three steps HEAT-BY-HEAT-EXCHANGER, TRANSFER, and DISTILL. Details of three of the primitive action descriptions are shown in Figures 5 and 6.[6] (The notation "C:substance" refers to a variable C restricted to be of type substance.) We have chosen an extremely simple (and obviously unrealistic) example because it allows us to focus solely on the limitations in plan recognition and design that we wish to address.

[6]We are being very loose in the representation of our actions here. We intend to use a more precise formalism such as the one developed by Schmolze (1987) for naive physics.

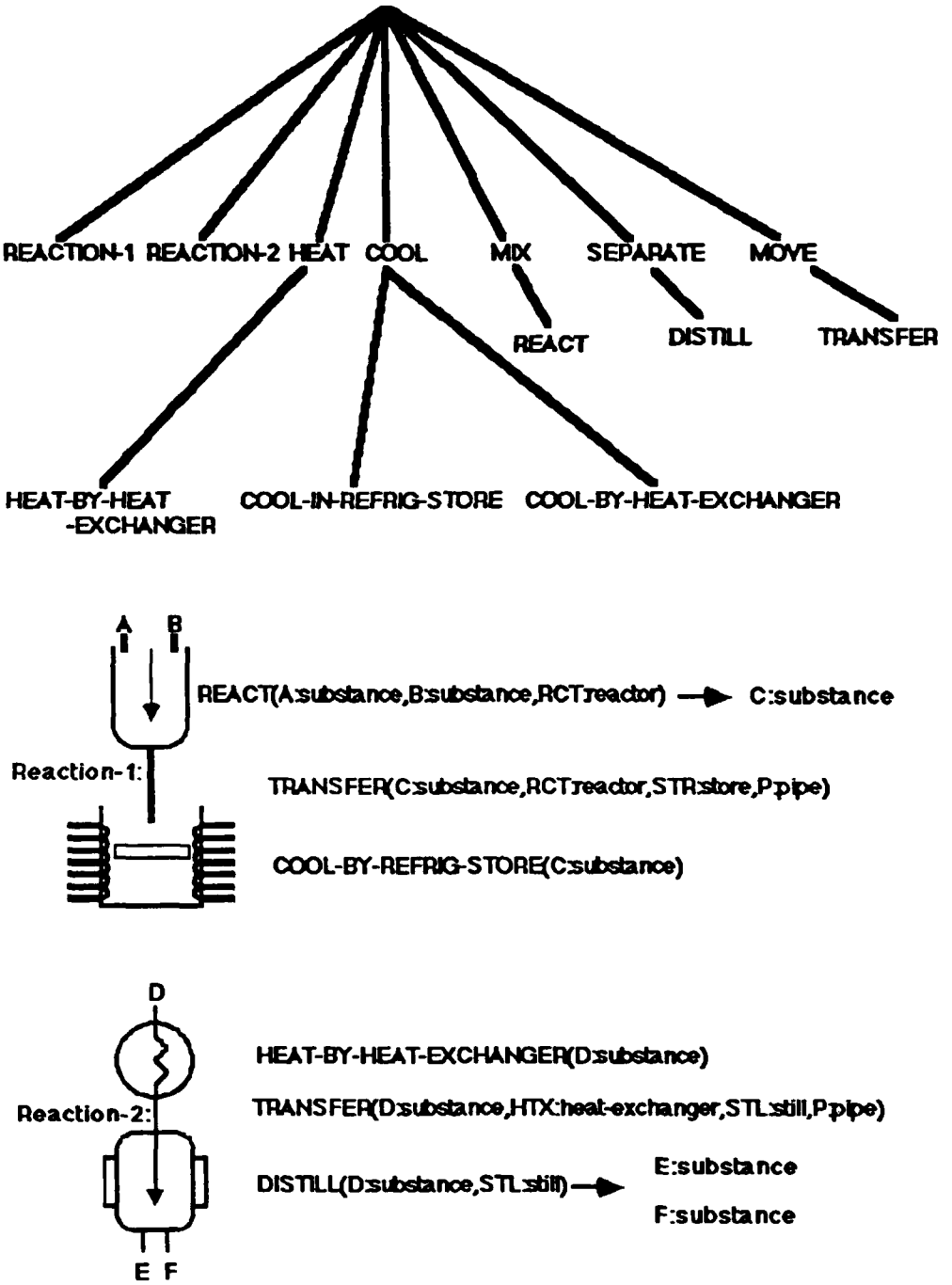


Figure 4: The design library: an action and decomposition hierarchy

```
COOL-IN-REFRIG-STORE(c:substance,t:time)
 precondition:
   volume(c:substance,t1:time)<volume(refrig-store,t1:time)
   not-in-refrig-store(c:substance,t1:time)
 effect:
   temperature(c:substance,t1:time)
          >temperature(c:substance,t2:time)
   not-in-refrig-store(c:substance,t3:time),
          t1<t2<t3
 constraint:
   in-refrig-store(c:substance,t2), t1<t2<t3

COOL-BY-HEAT-EXCHANGER(c:substance,t:time)
 precondition:
   not-in-heat-exchanger(c:substance,t1:time)
 effect:
   temperature(c:substance,t1:time)
          >temperature(c:substance,t2:time)
   not-in-heat-exchanger(c:substance,t3:time),
          t1<t2<t3
 constraint:
   in-heat-exchanger(c:substance,t2:time), t1<t2<t3
```

Figure 5: Defining the cooling process

```
DISTILL(c:substance,stl:still,t:time)
 precondition:
          input(stl:still,t1:time)=c:substance
          liquid-mixture(input(stl:still,t1:time))
          heated(input(stl:still,t1:time))
 effect:
          output(stl:still,t2:time)=components(input(stl:still,t1:time)),
                t1<t2.
```

Figure 6: Defining the distillation process

First, let us assume that this design library is complete, and that the user designs an instantiation of REACTION-1 by first introducing the REACT (by placing a reactor icon on

the display), followed by the introduction of COOL-IN-REFRIG-STORE (by placing a refrigerated store icon on the display), and then performs a TRANSFER (by connecting a pipe between the reactor icon and the refrigerated store icon). This design can easily be "parsed" using the "grammar" of the design library, in the manner of existing plan recognition systems. Furthermore, the recognized plan can then be used to enhance the design interface. For example, once the plan recognizer has unambiguously inferred the user's design, the system can use the design library to finish the remaining design details.

Now, however, let us remove the assumption that the system's design library is complete. That is, although the system and the user have the same repertoire of primitive actions, the system does not have complete knowledge of how these primitive actions can be combined. This allows a user to have a plan in mind that the system has never seen before (which is in fact what one would expect out of a CAD system). For example, assume the user is designing REACTION-3, consisting of a REACT, followed by a COOL-BY-HEAT-EXCHANGER, and then a TRANSFER. In this case the plan recognizer must be able to recognize a semantically (rather than syntactically) coherent plan. As discussed above, by incorporating existing as well as new techniques of plan generation and plan modification, CPR can use its incomplete planning knowledge towards this end. Consider a few simple examples:

• Plan Acquisition by Plan Modification

After observing the REACT, a traditional plan parser would postulate that REACTION-1 is being designed; after observing COOL-BY-HEAT-EXCHANGER the process would fail. CPR, however, could use the failed expectation, COOL-IN-REFRIG-STORE, to help recognize the input sequence as a coherent plan. In particular, CPR can postulate

substituting the primitive action COOL-BY-HEAT-EXCHANGER for COOL-IN-REFRIG-STORE in REACTION-1, since the effects of COOL-BY-HEAT-EXCHANGER and COOL-IN-REFRIG-STORE are similar and they both are instances of COOL. The design system could then use REACTION-1 in the design library as a template and suggest to the designer the TRANSFER between the reactor and the refrigerated store. The new sequence can then be stored as, say, REACTION-3. Hence, we recognize the design not by first principles but by modification of an already know design (Alterman 1986, Broverman 1987, Hammond 1987, Darpa 1988).

• Plan Acquisition by Plan Generation

Now suppose REACTION-1 is not in the decomposition hierarchy. In this case, CPR can only use local knowledge about actions to determine if the design REACT/COOL-BY-HEAT-EXCHANGER/TRANSFER is purposeful. For example, if the effects of the react achieve the preconditions of the cooling, the combination can be seen as purposeful, due to causal connection, and the sequence acquired. In contrast, if the effects violate the preconditions, the combination is not purposeful and plan relaxation is called (as below).

Note that we need not just consider consecutive actions. For example, suppose the effects of REACT neither violate nor achieve the preconditions of COOL-BY-HEAT-EXCHANGER. At this point, CPR can not say anything definitive other than that the two actions are neither incompatible nor compatible with each other. Instead, a chaining of preconditions and effects through potential action sequences is needed to see if the current action could be consistent with previous ones.

Notice that the design system can, even in the two cases illustrated here, provide advice to the designer by suggesting likely next additions to the design. For example, after viewing the REACT followed by COOL-BY-HEAT-EXCHANGER, it could suggest the TRANSFER action by drawing dimly a pipe between the output of the reactor and the input of the heat exchanger. The designer could accept the suggestion or provide his own.

• Non-Acquisition and Plan Relaxation

Finally, suppose we observe a COOL-BY-HEAT-EXCHANGER of substance C followed by a COOL-IN-REFRIG-STORE of the same substance. Plan generation knowledge would lead the system to determine that the proposed action sequence was redundant since the second COOL does not change the state of the world. Undoing a previous action can also be invalid. Suppose a MIX of two substances A and B to form another substance C is followed by a SEPARATE of C into its component parts A and B. That, too, would appear non-purposeful.[7] In these cases CPR postulates that miscommunication has occurred and calls the plan relaxation component to determine what was meant. The design system, hence, has become a partner in the design process alerting the designer to possible mistakes and suggesting how to recover.

4.3.3 An Example - Designing a Butane Isomerization Process

To give a larger perspective of the system we are proposing, we consider here a more complete example. Butane isomerization is a chemical process for changing the normal

[7]In a more realistic action hierarchy, such an undoing of a previous step isn't necessarily invalid. For example, in a chemical plant, a solid substance might be mixed with a liquid substance so that it can be transported through a pipe (such as a slurry of chopped coal mixed with water); the solid and liquid substances are then separated.

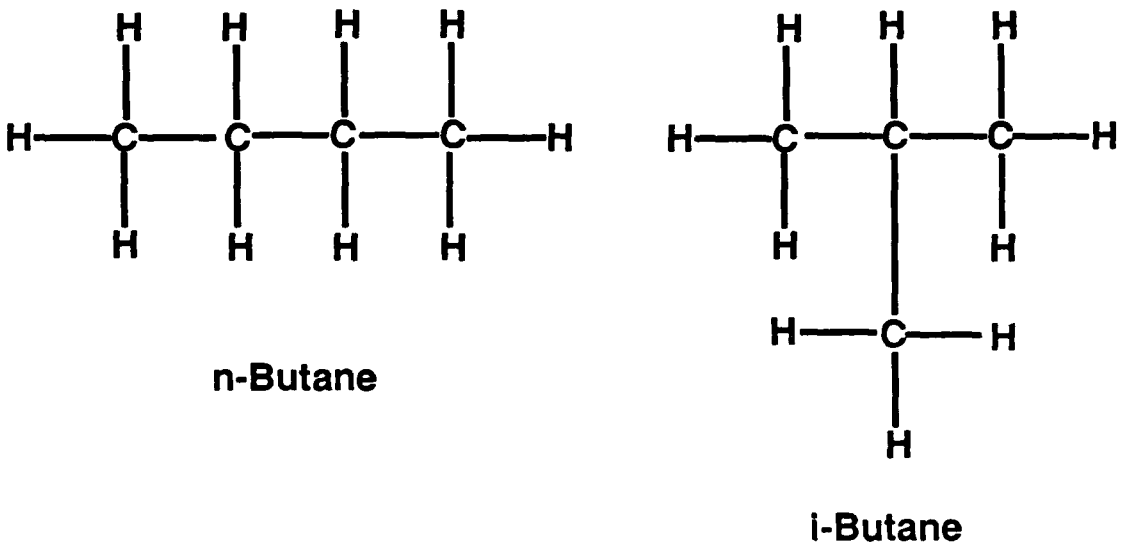


Figure 7: Butane isomers

configuration of *n*-butane (C_4H_{10}) to its isomer form *i*-butane (see Figure 7). The isomerization of butane can be accomplished commercially by reacting *n*-pentane with *n*-butane and separating out the resulting *i*-butane and the reaction by-products in a distillation column (hereafter called a "still"). The standard process for achieving this is shown in Figure 8. Note that any unchanged *n*-butane is separated from the reacted mixture by the still and recycled into the feed of the reactor. To avoid enriching the inert *n*-pentane, some of the recycling steam is released. We assume that the quantity of catalyst fed into the reactor is constant and that the still is set up to the prescribed parameters for performing the separation.

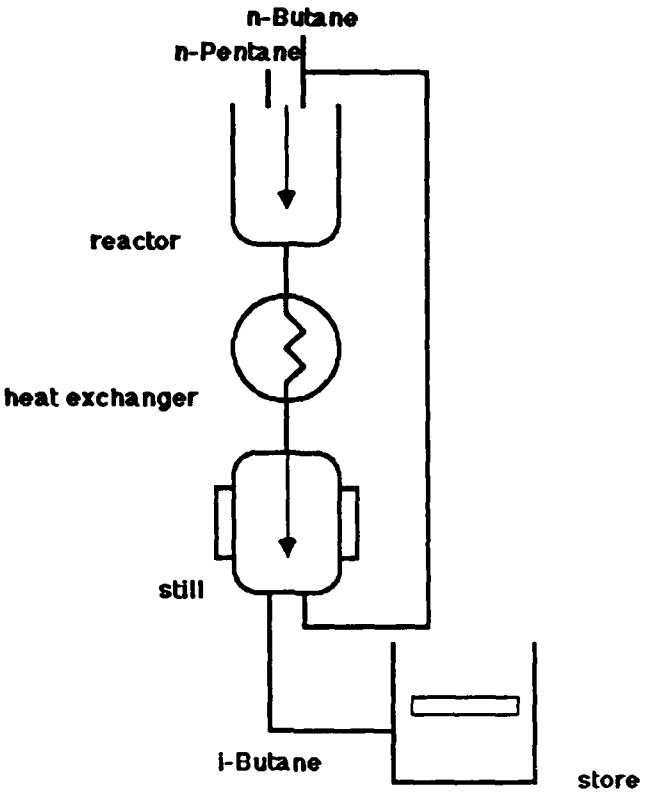


Figure 8: Butane isomerization

We assume for the sake of our example that our design library contains only the butane isomerization design shown in Figure 8. The user of our proposed plan-based computer aided design system (which we call "CHECS" for CHemical Engineering CAD System) selects one component at a time, adding them to the display of the evolving design, and selects graphical actions one at a time, to perform on the components. Figure 9 provides an illustration of a simple chemical process design system. In the figure, the designer has already placed three components on the screen - two "Feed"s and one "Reactor." In Figure 10, the user adds a "Pipe" between an output of one of the "Feed"s and an input to the "Reactor" by selecting "Add-Pipe" from the menu and indicating with the mouse the beginning and end points of the pipe. CPR tracks the user's selected actions, noting both

the graphics-level actions (e.g., Add-Icon, Remove-Icon, Move-Icon, etc.) and their corresponding chemical process actions (e.g., Add-Reactor, Add-Pipe, etc.) taken. It updates the effects of an action (and any implications thereof) and will contrast them against the preconditions of the next user action as well as the effects of previous actions. It compares the design so far with those in the design library.

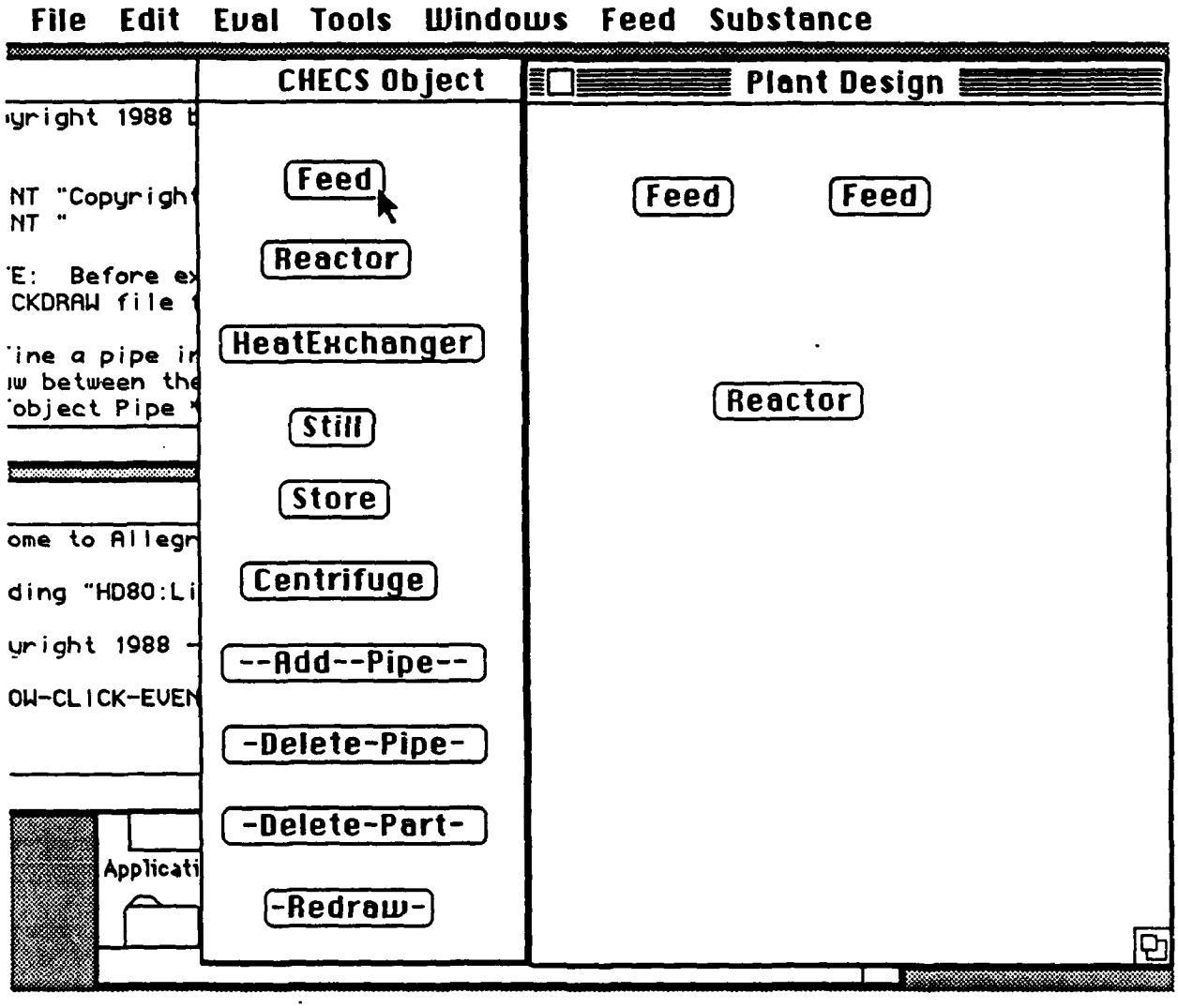


Figure 9: Adding components to the design

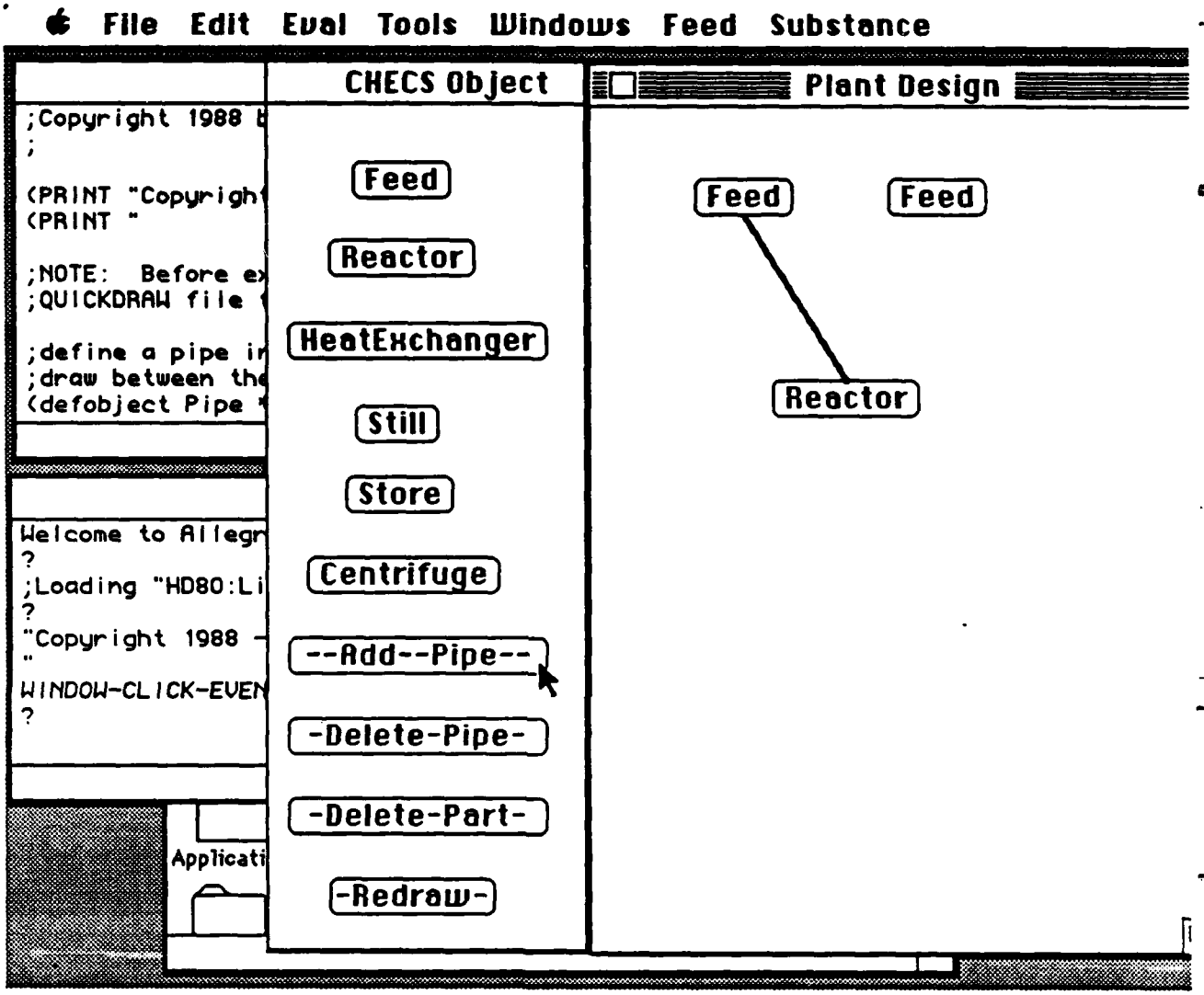


Figure 10: Adding a pipe between components

Now, suppose our designer knows that he already has in his stock of components a "heated reactor" (a reactor that comes with built in heating coils). To save costs, he applies a different perspective on butane isomerization. He knows that a "Heated Reactor" can provide him with the same results as placing a "Heat Exchanger" on the output of a standard "Reactor." In Figure 11, he specifies the "Reactor" to be of type "Heated Reactor"

by double-clicking on the reactor icon to yield a menu of reactor types and then selecting the entry for "Heated Reactor." At this point, the user's design conflicts with those in the design library (in the example here, the one design shown in Figure 8). The effects, however, of his addition of the "Heated Reactor" match those formed by the connection of the "Pipe" to the "Reactor" and "Heat Exchanger" in the design in the library. Hence, the system views the new design as being both purposeful and novel (at least to this point in the design process).[8] It can suggest the addition of the "Still" and the "Store" now that it has acquired a match against the butane isomerization design in the design library. The resulting design is shown in Figure 12. If at the end of the design process all the preconditions and effects are not connected (for example, if the designer did not provide "heat" to the input to the "Still"), the design cannot be seen as purposeful. In that case the design system interrupts the designer and points out design errors from its causal perspective (e.g., the required "heating" action did not occur).

4.4 Summary

We propose that a more constructive view of plan recognition is needed to provide robustness to plan-based systems. In particular, plan recognition must contend with a fact about the world: agents have incomplete knowledge of each other and of the world. However, if we drop the completeness assumption, previous plan recognition technology no longer suffices. We have proposed a framework, CPR, that is designed to survive without the completeness assumption. CPR supports the recognition of novel user plans and the detection and correction of miscommunication in user plans. By adding "semantic"

[8]Notice that it would have been a more difficult task if the design library contained the butane isomerization design containing the heated reactor but not the one using the separate reactor and heat exchanger. The plan recognizer would have to recognize that the effects of the heated reactor were now spread between two components, the reactor and the heat exchanger.

knowledge to the plan recognition process, CPR can extend its syntactic knowledge. While this extension introduces its own limitations (e.g., the construction operators expand the search space), we can avoid these limitations by providing assistance locally and only popping to a more global level when the designer has provided enough information to elucidate his goals.

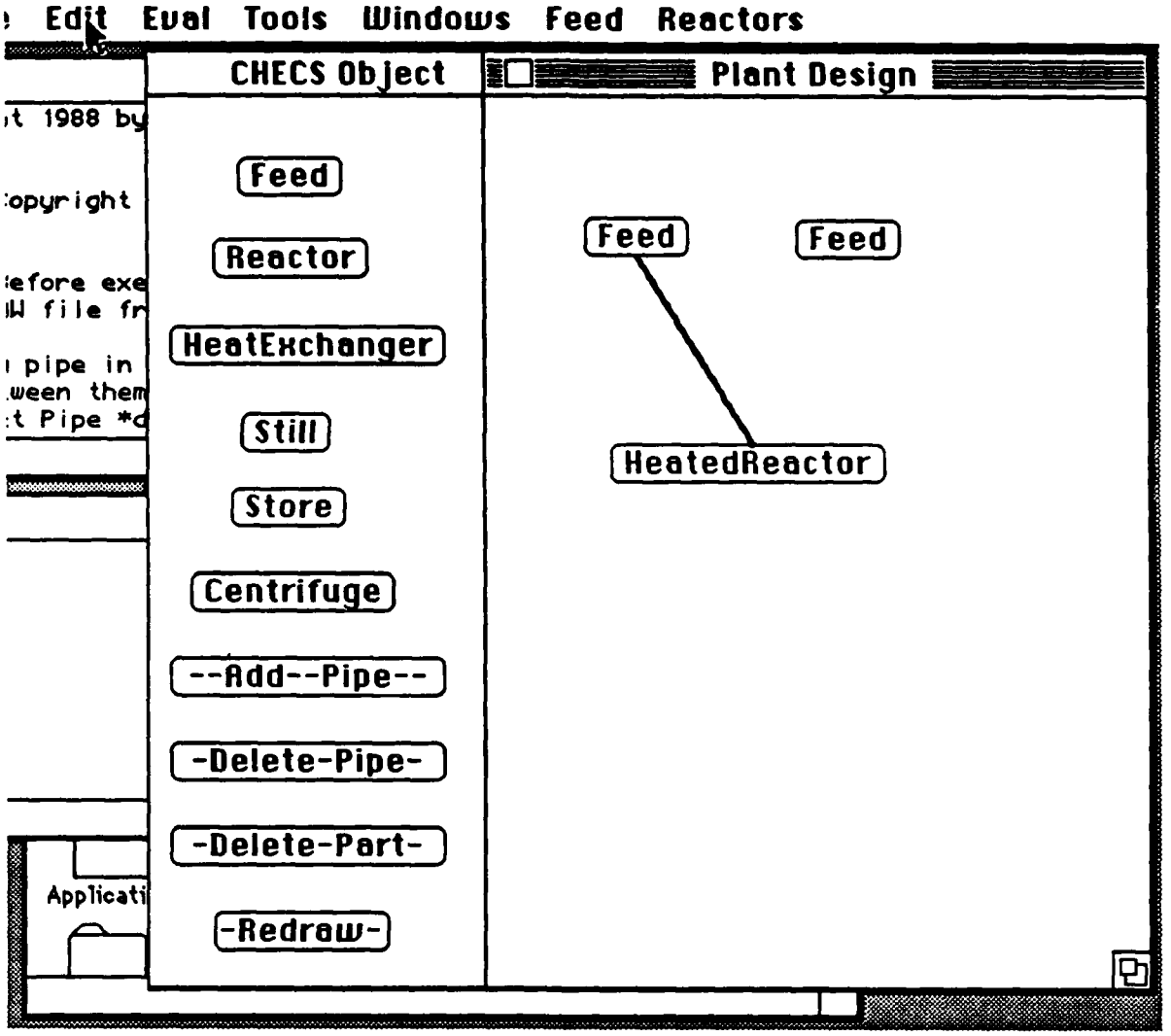


Figure 11: A novel approach to butane isomerization

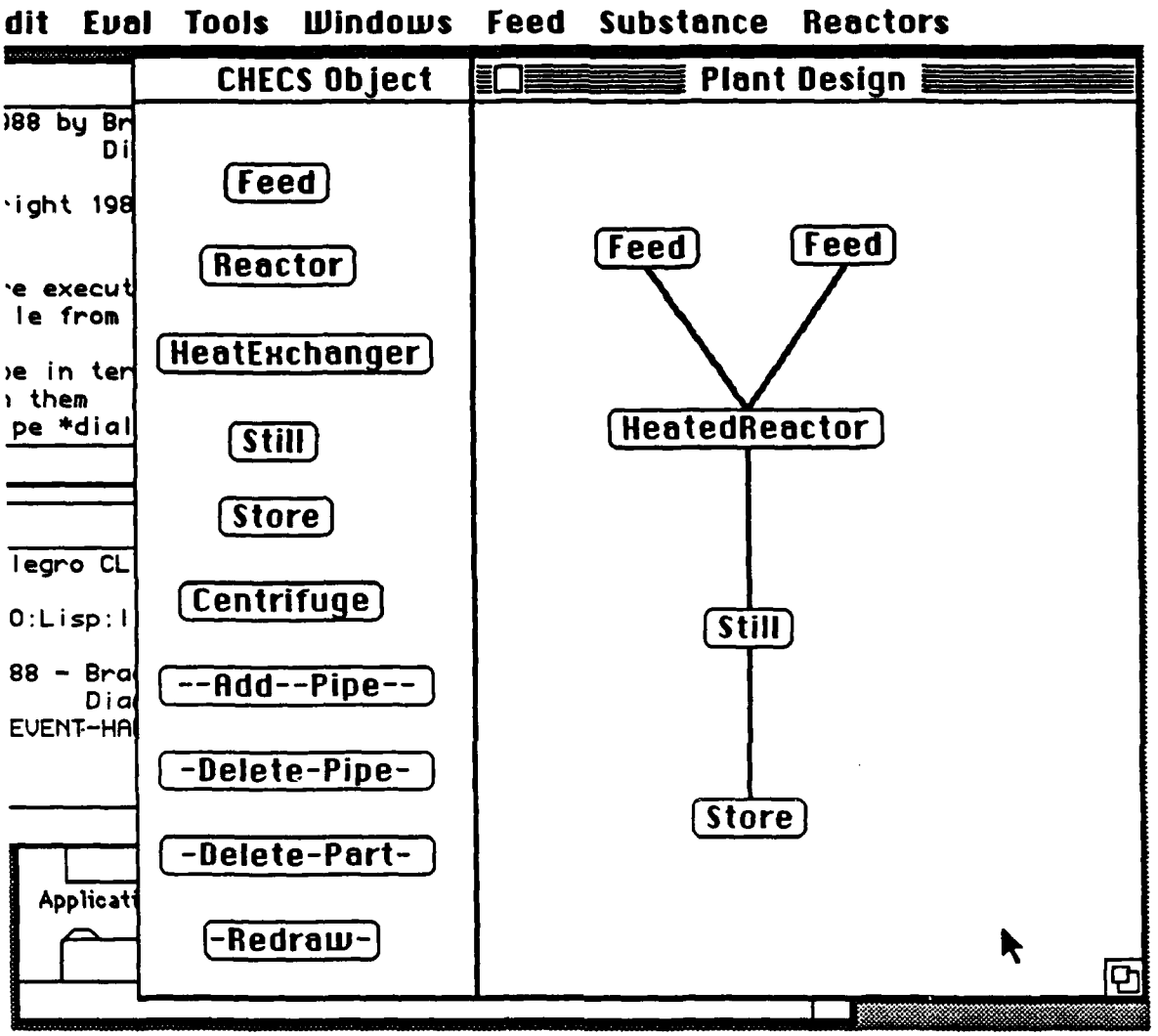


Figure 12: The completed novel butane isomerization design

Computer aided design can benefit greatly from the addition of an intelligent interface. Plan recognition can provide context-sensitive actions to the interface technology to make it more reactive to user's needs. Constructive plan recognition provides a boost to CAD by

promoting its status to a helpful partner in the design process. CAD, supplemented by the knowledge gained by plan recognition, can recognize novel designs, detect mistakes, and give suggestions.

References

Allen, James F. *A Plan-Based Approach to Speech Act Recognition.* PhD thesis, University of Toronto, 1979.

Allen, J.F., and Perrault, C.R. Analyzing intention in utterances. *Artificial Intelligence* 15(3):143-178, 1980.

Alterman, Richard. An Adaptive Planner. In *Proceedings of AAAI-86*, pages 65-69. Philadelphia, Pa., August, 1986.

Broverman, Carol A. and W. Bruce Croft. Reasoning About Exceptions During Plan Execution Monitoring. In *Proceedings of AAAI-87*, pages 190-195. Seattle, Wa., July, 1987.

Brown, David C. and B. Chandrasekaran. Knowledge and Control for a Mechanical Design Expert System. *Computer* 19(7):92-100, July, 1986.

Bruce, Bertram. Robot Plans and Human Plans: Implications for Models of Communication. In I. and M. Gopnik (editors), *From Models to Modules: Studies in Cognitive Sciences from the McGill Workshops*, pages 97-114. Ablex, Hillsdale, New Jersey, in press.

Carberry, Mary Sandra. *Pragmatic Modeling in Information System Interfaces.* PhD thesis, University of Delaware, 1985.

Carberry, Sandra. User Models: The Problem of Disparity. In *Proceedings of COLING-86*, pages 29-34. Bonn, West Germany, August, 1986.

Carver, N.F., Lesser, V.R., and McCue, D.L. Focusing in Plan Recognition. In *Proceedings of the National Conference on Artificial Intelligence*, The American Association for Artificial Intelligence, Austin, TX, 1984.

Cohen, Philip R. *On Knowing What to Say: Planning Speech Acts*. PhD thesis, University of Toronto, 1978.

Defense Advanced Research Projects Agency. *Proceedings of the DARPA Workshop on Case-Based Reasoning*, 1988.

Fikes, R. E. and Nilsson, N. J. A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189-208, 1971.

Genesereth, Michael R. The Role of Plans in Automated Consultation. In *Proceedings of IJCAI-79*, pages 311-319. Tokyo, Japan, August, 1979.

Goodman, Bradley A. Reference Identification and Reference Identification Failures. *Computational Linguistics* 12(4):273-305, October-December, 1986.

Grosz, B.J. and Sidner, C.L. Plans in Discourse. In P. Cohen and M. Pollack (editors), *Discourse, Communication and Intention*. Massachusetts Institute of Technology Press, Cambridge, Ma., in press.

Hammond, Kristian J. Explaining and Repairing Plans that Fail. In *Proceedings of IJCAI-87*, pages 109-114. Milain, Italy, August, 1987.

Huff, Karen and Victor Lesser. *Knowledge-Based Command Understanding: An Example for the Software Development Environment*. Report TR 82-6, Computer and Information Sciences, University of Massachusetts at Amherst, Amherst, Ma., 1982.

Kass, A. Modifying Explanations to Understand Stories. In *Proceedings of the Cognitive Science Society*, pages 691-696. Amherst, Ma., 1986.

Kautz, Henry A. and Allen, James F. Generalized Plan Recognition. In *Proceedings of AAAI-86*, pages 32-37. Philadelphia, Pa., August, 1986.

Kautz, Henry A. *A Formal Theory of Plan Recognition.* PhD thesis, University of Rochester, 1987. Also, TR215, University of Rochester, Dept. of Computer Science, Rochester, N.Y

Kolodner, Janet L., Robert L. Simpson, Jr., and Katia Sycara-Cyranski. A Process Model of Case-Based Reasoning in Problem Solving. In *Proceedings of IJCAI-85*, pages 284-290. Los Angeles, August, 1985.

Koton, Phylis. Model-Based Diagnostic Reasoning Using Past Experience. Talk in the BBN Laboratories AI Seminar Series, BBN Laboratories Inc., Cambridge, Ma., June 14, 1988

Litman, Diane J. and James F. Allen. A Plan Recognition Model for Subdialogues in Conversations. *Cognitive Science* 11:163-200, 1987.

Mittal, Sanjay, Clive L. Dym, and Mahesh Morjaria. PRIDE: An Expert System for the Design of Paper Handling Systems. *Computer* 19(7):102-114, July, 1986.

Pollack, Martha E. *Inferring Domain Plans in Question-Answering*. PhD thesis, University of Pennsylvania, 1986. Also, Report MS-CS-86-40 of the Department of Computer and Information Science, University of Pennsylvania.

Pollack, Martha E. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Proceedings of the 24th Annual Meeting of the Association of Computational Linguistics*, pages 207-214. Columbia University, New York, N.Y., June, 1986.

Sacerdoti, Earl D. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence* 5:115-135, 1974.

Samad, Tariq. *A Natural Language Interface for Computer-Aided Design.* Kluwer Academic Publishers, Boston, 1986.

Schmidt, C. F., Sridharan, N. S., and Goodson, J. L. The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence. *Artificial Intelligence* 11:45-83, 1978.

Schmolze, James. *Physics for Robots.* Technical Report 6222, BBN Laboratories Inc., Cambridge, Ma., September, 1987.

Sidner, Candace L. Plan parsing for intended response recognition in discourse. *Computational Intelligence* 1(1):1-10, 1985.

Tong, Christopher. AI in engineering design. *Artificial Intelligence in Engineering* 2(3):130-132, 1987.

Tong, Christopher. Toward an engineering science of knowledge-based design. *Artificial Intelligence in Engineering* 2(3):133-166, 1987.

Wilensky, Robert. *Planning and Understanding.* Addison-Wesley Publishing Company, Reading, Ma., 1983.

Woods, W.A. Cascaded ATN Grammars. *Amer. J. Computational Linguistics* 6(1):1-15, Jan.-Mar., 1980.

5. Distributed Know-How and Acting: Research On Collaborative Planning[1]

Barbara J. Grosz, Harvard University

Candace L. Sidner, BBN Systems and Technologies Corporation

5.1 Introduction

Our research project is aimed at the construction of an intelligent agent that can collaborate with other agents in the design and performance of plans to achieve shared objectives. Collaborative planning and action are required for a range of tasks involving multiple agents, including such undertakings as artifact construction (e.g., assembly tasks, building construction, and contracting), financial management, and team activities (e.g., moving equipment, search and rescue management, musical performances).

Although substantial research has been undertaken in the areas of multi-agent planning systems [Geo87a,Ros82,Mor86], none of this work provides for collaboration in the planning process. In particular, the research presumes that the plans of multiple agents can be defined in terms of the private plans of individual agents. This work has assumed that all coordination of the actions of different agents can be done by one central master or that the individuals can form their own plans completely before coordinating with other agents.

Research under the first assumption has articulated an architecture for multi-agent planning in which a central agent creates plans that are then given to a set of agents to carry out; the

[1]Several portions of this paper are taken from [GS88].

central agent has or must obtain all the information relevant to the plan construction, and does not rely on another agent to play a role in the planning process. Under the second assumption each agent's plans are private. There is a need to share information only when an agent recognizes that some part of its plan will need to be coordinated with what it assumes is the plan of the other agent. Should that coordination be needed, the two agents will interact to establish the coordination point.

Although this research has addressed key issues in a number of areas (e.g., providing formal techniques for reasoning about the beliefs of multiple agents and for coordinating [externally] the actions of multiple agents [Kon84,Moo85], the design of methods for handling conflicts among the desires of different agents [Ros82,Wil83], and providing techniques for synchronizing actions of multiple agents to avoid harmful interactions), the basic models of plans are insufficient to support collaborative planning. Algorithms for plan recognition (which has been shown to be crucial to discourse participation [Bru75,SSG79,All79,Sid83,KA86]) are likewise not sufficient to support recognition in the context of a collaborative plan.

Collaboration occurs in two ways in planning: (1) the planning process itself may be collaborative, or (2) the plan agents come to have may involve collaborative activity.[2] In the planning process, any aspect of a plan may be a subject of negotiation and collaborative consideration. The agents may make together such decisions as what goal they wish to achieve, what actions performed by each will achieve that goal, and when they will do each action. To make these decisions requires sharing information that may be known only to one agent. Negotiation is required to determine goals that are of mutual value and to reconcile disagreements when goals are not. Participation thus requires communication.

[2]We use the term plan for a collection of beliefs and intentions as in Pollack's [Pol86] and our own previous work [GS88]. We use recipe (or blueprint) for a set of actions, performed by any agent, that lead to a desired goal.

Many acts are not under the sole control of one agent, and hence interactive participation by all agents is required.

The outcome of this dynamic planning process is a plan held by the agents. When the plan is collaborative, it is a configuration of mutual beliefs about the actions to be performed and relations among those actions, and of the intentions of the two agents to perform actions. When the plan is to be performed by one agent, the plan is a configuration of that agent's beliefs and intentions about the action and action relationships; the two agents also share mutual beliefs, but only about the one agent's plan.

Serious consideration of dialogue makes it clear that an assumption that one agent is in control is the wrong basis on which to build a theory of planning and acting. We call any such assumption the master-slave assumption. This assumption encourages theories that are unduly oriented toward there being one controlling agent and one reactive agent. Only one agent has any control over the formation of the plan; the reactive agent is involved only in execution of the plan (though to do so he must first figure out what that plan is). We conjecture that the focus of speech act and plan recognition work on single exchanges underlies its (implicit) adoption of the master-slave assumption. To account for extended sequences of utterances, it is necessary to realize that two agents may develop a plan together rather than merely execute the existing plan of one of them. That is, language use is more accurately characterized as a collaborative behavior of multiple active participants.

Language use is not the only form of cooperative behavior which requires a notion of collaborative plans. A variety of nonlinguistic actions and plans cannot be explained solely in terms of the private plans of individual agents (cf. Searle[Sea88]). For example, consider the situation portrayed in Figure 1. Two children each have a pile of blocks; one

child's blocks are blue, the other's green. The children decide to build together a tower of blue and green blocks. It is not the case that their plan to build this tower is any combination of the first child's plan to build a tower of blue blocks with some empty spaces (in just the right places to match the other child's plan) and the second child's plan to build a tower of green blocks with some empty spaces (again in just the right places). Rather, they have a collaborative plan that includes actions by each of them (the first child adding blue blocks, the second green ones). In a more practical vein, the concept of collaborative plans provides a foundation for theories of collaborative behavior that could provide for more flexible and fluent interactions between computer systems and users undertaking joint problem-solving activities (e.g., systems for diagnosis).

In this paper we discuss characteristics of the collaborative planning process, define collaborative plans involving a number types of different actions and action relationships and present evidence for the collaborative view from videotapes of agents planning and performing a complex task.

5.2 The Collaborative Planning Process

Collaboration occurs in planning situations where more than one agent is required to produce and act on joint plans that satisfy their common goals. The collaborative planning process can be characterized by three basic features: negotiation of common goals and of the appropriate sequence of actions to achieve the goals, limited knowledge of each participant, and distributed potential for action.

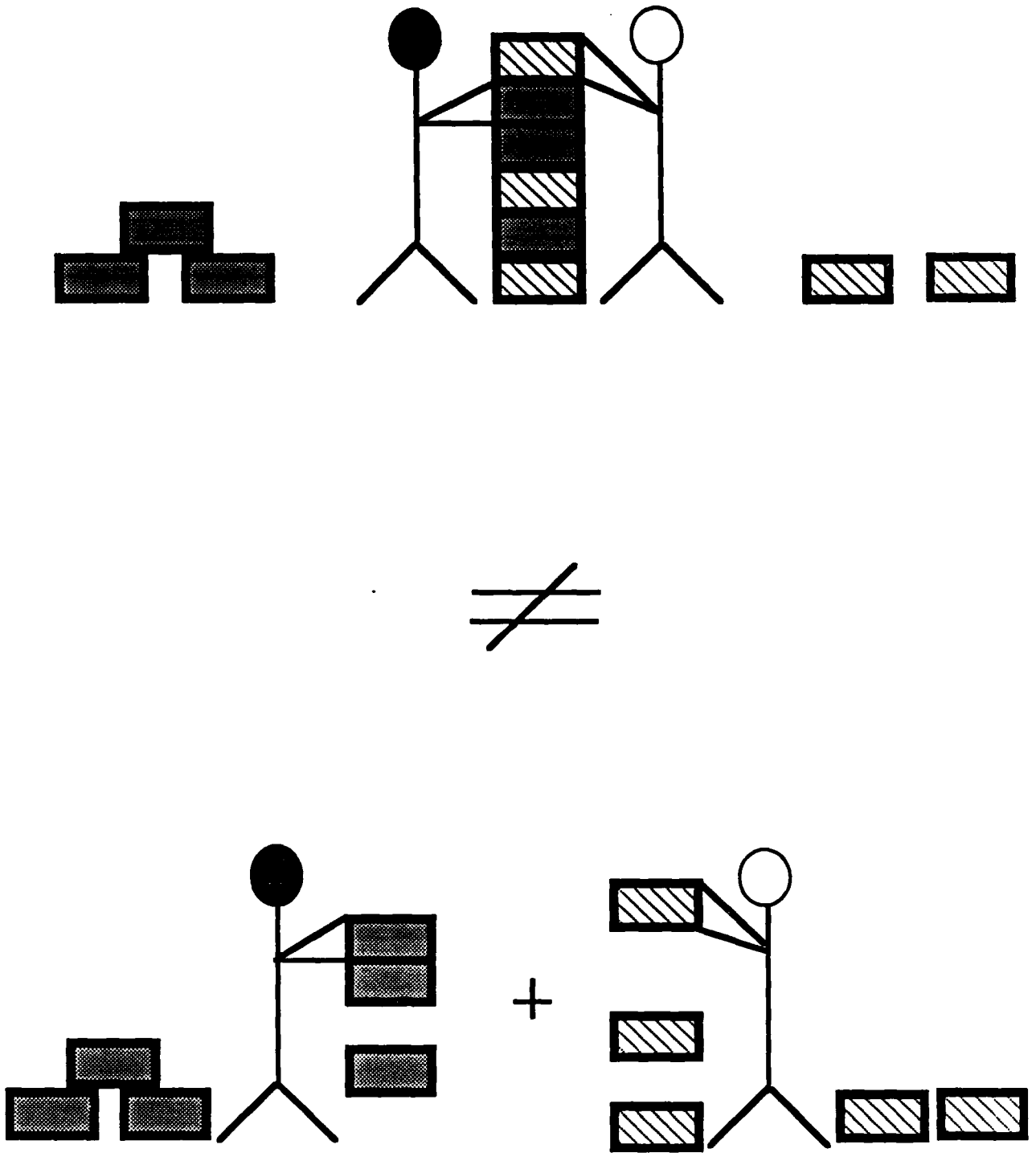


Figure 1: A Collaborative Block Building Example

The goals of a plan need not be the choice of only one of the agents. Rather, one agent may have a need that others join in satisfying, or several agents may together identify common goals. They must necessarily negotiate in order to determine which of their goals are to be attempted (when not all can be), and in which order they might actually be satisfied (when more than one is undertaken). When negotiating a common goal, agents are not guaranteed that they will succeed in reaching accord. Hence, establishing the common goal is a part of the collaborative planning process.

Plans that require a common goal necessarily include collaboration; no one agent can solely determine the goal because agents may lack information critical to knowing what the common goal is. The goal becomes common in part by sharing information about individual goals with other agents. In addition agents lack the power to determine agreement for other agents. By this we mean that each agent can choose to assent or dissent to a proposed common goal. No agent can do this for another (without prior approval, a matter that can be quite complicated in its own right), and agents who are not free to agree or disagree are not able to collaborate. Thus collaboration does not occur when one agent determines another's goals without consent of that agent. For example, a slave does not participate with his or her master in a collaborative negotiation of a goal since the slave is not permitted to disagree.

Typically no single agent has all the knowledge needed to produce a plan that satisfies the common goal. Each agent contributes different know-how to the planning process by communicating to others of some of their individual, relevant knowledge. Typically they share information such as knowledge of the circumstances in which the actions will take place, of beliefs about the reasons for failures of previous plans to succeed and of

knowledge about the actions that each agent can perform.

The circumstance of partial information on the part of planning agents can motivate agents to participate in planning collaboratively (as well as, we conjecture, in seeking information through activities such as spying when they wish not to collaborate). Information is partial in the large when agents lack knowledge of each other's knowledge; it is partial in the small for any single agent who lacks knowledge or belief about his own internal state. Partial information in the large characterizes most of the planning process devoted to sharing individual knowledge about the circumstances, errors and actions mentioned above. While partial information in the small may seem a contradiction in terms, it is not. Many times a human agent will inform others that he is unaware of what he believes or knows with respect to an action or desire; he often requests time in which to determine just what it is that he believes.

Just as know-how is distributed among the agents, so is the potential for action. No single agent may be able to perform all the actions called for in the plans. Rather, each agent must determine what she can contribute to the collaborative planning process and what actions she can undertake that will accomplish part (or all) of the common goal. In addition to determining one's own possible contributions, an agent must negotiate collaboratively about (1) what actions will bring about the desired outcome; (2) which agent will perform each of the actions that are agreed to. Just as with common goals, the collaboration in this negotiation arises from the partial information distributed among the agents and from the individual right to assent or dissent. Negotiation is needed because no one agent is in a position to decide for other agents what they will do, and because the agents may have differing beliefs about what actions will bring about the desired outcome. Hence each agent must individually contribute some of its private knowledge and may form individual

intentions that are appropriate to the overall plan configuration.

Collaborative planning is a cyclic process, not a one-shot deal. The cycle is evident in the process of deciding what action will be done and who will do it, because the two matters are interdependent. What action is chosen often depends on who can do it as well as what act, in the abstract, might lead to the desired outcome. This cycle also illustrates that planning is interactive not simply because communication is interactive but also because of the interdependence among the kinds of information needed to make a decision.

The other evident cycle in planning is the planning and acting cycle. Nilsson et al. [NCRF] suggest that collaborating agents plan a part of a task and then do it. Our data support the intuition that this is often the case. In fact agents can often be uncertain of how they will reach the overall (end) goal, and can proceed with planning a sub-part of what they believe is on the way to the goal. They do so not simply to behave in a cleanly modular way. They must plan and act on the partial information they have about the world and the final outcome since complete information often is not available.

5.3 Shared Plans

We have defined a construct, SharedPlan of multiple agents, that summarizes the agents' holding of certain beliefs and intentions when they act collaboratively [GS88]. A SharedPlan accounts in part for the collaborative behavior we believe is manifest in most distributive planning and acting circumstances. The definition is based on Pollack's definition of a single agent having a simple plan.[3]. Like that work, our definition differs

[3]This means for the moment we will only consider actions related by generation; we will discuss the extension to enabling relationships later.

from previous research in taking a plan to be a configuration of beliefs and intentions about actions and their executability rather than a data structure encoding a sequence of actions. This is not to say that what Pollack calls the recipe, i.e. the actions and their interrelationships are not important; beliefs about these actions and relationships are a central element of the plan. Rather the recipe alone is not sufficient. It is only a part of what is established in planning and in the plan that the agents come to hold.

Our definition of a SharedPlan departs significantly from previous research in providing a means of specifying a plan that involves action by several agents as well as the associated beliefs and intentions. The definition provides for (1) the inclusion of action relations among actions performed by more than one agent, and (2) the introduction of mutual beliefs that the agents must hold. In particular two agents, G1 and G2, will be said to have a SharedPlan to perform an action A just in case they have the following beliefs and intentions:

SharedPlan (G1 G2 A) $\Leftrightarrow$

1. $MB(G1\ G2\ (EXEC\ (\alpha_j, G_i)))$
2. $MB(.....)$
3. $MB(G\ 1\ G\ 2(INT\ (G_i, \alpha_j)))$
4. $MB(G1\ G2(INT\ G_i, BY(\alpha_j, A)))$
5. $INT(G_i, \alpha_j)$
6. $INT(G_i, BY\ (\alpha_j, A))$

The index j ranges over all of the acts involved in doing A; for each α_j one of the agents, G1 or G2, is the agent of that action. That is, the action consisting of the act-type α_j ,

done by agent G1 or G2 (as appropriate), at time t contributes to G1 and G2's plan to accomplish A. Like Pollack, we use the constructor function ACHIEVE to turn properties (i.e., states of affairs) into act-types. If G1 and G2 construct a SharedPlan to have a clean room, we will say there is a SharedPlan(G1 G2 Achieve(Clean-room)).

The content of clause (2) depends on the types of actions being done. So far we have developed definitions of three classes of SharedPlans: they involve simultaneous actions by two agents, conjoined actions by two agents and sequence of actions by two agents.[4]. In this paper we will discuss each.[5]

5.3.1 Simultaneous Action

Simultaneous actions are an example of the kind of activity that requires collaborative planning. They form the basis of one type of SharedPlan. To obtain the result of simultaneous actions (a condition in the world) requires actions on the part of both agents at the same time. A simple example of such an action is lifting a heavy object. For there to be a shared plan involving simultaneous action, the agents involved must hold the following mutual belief:

$$MB(G1, G2, [OCCURS(\alpha_i, G1, T1) \Leftrightarrow GEN(\beta_j, \gamma, G2, T1) \;\&\; OCCURS(\beta_j, G2, T1) \Leftrightarrow GEN(\alpha_i, \gamma, G1, T1)]\, T0)$$

This formula stipulates that the participants believe that the performance of action β_j by G2 at time T1 GENerate (informally, read as "lead to") the desired outcome γ when and only when there is an occurrence of action α_i by G1 also at time T1; in addition the

[4]Typically in a complex plan, several types of relations among actions will hold. The resulting version of clause (2) will be more complex than we will illustrate.

[5]We also have considered single agent plans with collaborative negotiation of the goal but we will not discuss them here.

performance of action α_1 by G1 at time T1 GENerate the desired outcome γ when and only when there is an occurrence of action β_1 by G2 at T1.[6] This formula records several features of the use of simultaneous actions. First, the appropriate simultaneity relation will hold only when actions by both agents produced the desired outcome. For if either agent's actions alone produce the desired outcome γ, then this formula will fail to hold. Second all actions must be undertaken at the same time (T1), an intuitively relevant condition of simultaneity. Third, each agent's actions must and can GENerate the desired outcome only in the presence of the other's actions; one agent's actions do not occur simply as a means of causing or enabling the actions of the other agent that generate γ.

For example, suppose G1 and G2 have a SharedPlan to pick up a large, heavy box. G1 is to pick up one end (α_1) and G2 the other (β_1) G1's action, undertaken when G2 is performing her action, will generate having picked up the box. This statement is insufficient to describe the conditions for successful simultaneous action; G2's action is more than a mere coincidence that happened when G1 started lifting. It is intentional and intended to work with G1's action. G2's action likewise generates picking up the box only when G1 is also undertaking the lifting. Hence, simultaneous actions achieve the desired goal only when the actions of both agents together lead to the outcome.

Since SharedPlans stipulate mutual belief between the agents, the conditions under which these beliefs come to be held is crucial to the SharedPlan. Typically agents communicate in order to bring about mutual belief; they also can rely on visual recognition when they are co-present and can see each other. It is not the case that all the mutual beliefs must be either

[6]Following Pollack [Pol86], we have adopted Allen's interval-based temporal logic [All84] as the basic formalism for representing actions. We use Pollack's modification of the predicate representing the occurence of an action, OCCURS: the predication OCCURS (α, G, t) is true if and only if the act-type α is performed by G during time interval t. We also adopt Pollack's use of Goldman's [Gol70] *generates* relation.

communicated or observed. In the constructed, very simple discourse below, which is taken from our previous paper [GS88], the utterances provide explicit mention of mutual beliefs of desires, and intentions, and of assent to the actions to be undertaken. There is no mention of the mutual belief about the GEN-simultaneous relation for lifting the piano. As we shall see, in this case, it can be inferred

Discourse D1:

1. S1: I want to lift the piano.
2. S2: OK.
3. I will pick up this [deictic to keyboard] end.
4. S1: OK
5. I will pick up this [deictic to foot] end.
6. S2: OK.
7. Ready?
8. S1: Ready.

We assume, as described in previous work, that the participants can infer from utterances (1), (3), and (5).

(1) MB(S1, S 2, Desire(S 1 lift(piano)))

(3') MB(S1, S 2, INT (S 2 lift(keyboard -end)))

(5') MB(S1, S 2, INT (S1 lift(foot -end))).

From (1'), rules of conversation, and appropriate assumptions about the agents' cooperativeness ilar type of SharedP(*vide*[GS88]), they can infer that

MB(S1, S2, Desire (S1, Achieve (SharedPlan1 (S1, S2, lift (piano)))))

Hence, following utterance (1), G2 could (coherently) respond in any one of the following ways:

- explicitly dissent from accepting the SharedPlan ("I can't help now."),
- implicitly dissent ("I hurt my back."),
- explicitly assent to construct a SharedPlan (above example),
- implicitly assent to construct a SharedPlan ("Which end should I get? Do you have a handtruck?").

In Utterance (2), S2 explicitly assents to work on achieving the SharedPlan for lifting the piano. Utterance (3), by providing the information in (3'), provides information needed for the SharedPlan. It expresses the intentions exhibited in clauses (3) and (4) of the SharedPlan, and implicitly expresses S2's belief that S2 can execute the intended action. S1's assent to this proposed action in utterance (4) allows derivation of mutual belief of executability as well as the relevance of this act to achieving the desired goal (i.e. a portion of the belief exhibited in clause (4)). Utterance (5), analogously to Utterance (3), expresses intentions (now additional ones) exhibited in clauses (3) and (4), as well as a new individual belief about executability. Utterance (6) allows derivation of mutual belief of executability.

This discourse does not include any explicit mention of the generation relationship exhibited in Clause (2). From the context in which (3) and (5) are uttered, the participants can infer that the mentioned actions are seen to participate in a generation relationship with the desired action. That these actions together are sufficient is implicit in utterances (7) and (8). S1 and S2 can now infer that the generation relation exhibited in Clause (2) holds.

Their SharedPlan comprises the following mutual beliefs and intentions:

SharedPlan1(S1 S2 lift[piano])

1. MB(S1 S2 (EXEC (lift(foot-end)) S1)) & (MB(S1 S2 (EXEC (lift(keyboard-end)) S2)))

2. MB(S1, S2, GEN-simultaneous[lift(foot-end) & lift(keyboard-end), lift(piano), S1 & S2])

3. MB(S1 S2 (INT S2 (lift(keyboard-end)))) & MB(S1 S2 (INT S1 (lift(foot-end))))

4. MB(S1 S2 (INT S2 (BY (lift(keyboard-end)) lift(piano)))) & MB(S1 S2 (INT S1(BY (lift(foot-end)) lift(piano))))

5. INT(S2 lift(keyboard-end)) & INT(S1 lift(foot-end))

6. INT(S2 (BY (lift(keyboard-end)) lift(piano))) & INT(S1 (BY (lift(foot-end)) lift(piano)))

The use of the concept of a SharedPlan eliminates the need for any notion of one agent intending for another agent to intend some action; i.e., we have no need for clauses of the form Intend(G1 Intend (G2 Do (Action))). Rather (as exhibited in Clause (2)), the participants must have mutual belief of the ways in which actions by each agent done simultaneously generate a single (joint) action [namely, lift(piano)]. As stated in clause (6), S2 intends to lift the piano by lifting the keyboard-end (alone); she can do this only because she believes (there is a mutual belief) that S1 will simultaneously lift the foot-end.

5.3.2 Conjoined Actions

A similar type of SharedPlan may be constructed when the actions of two agents taken

together, but not necessarily performed simultaneously, achieve a desired result. For example, a table may be set by two people each of whom performs some of the necessary actions (e.g., one putting on the silverware, the other the plates and glasses). In such cases, there is a simple conjunction of actions, rather than a need for simultaneity. That is, although the actions must all be performed within some time interval, say T_E, they need not be performed at exactly the same time. For this case, SharedPlan2(G1, G2, Achieve(conjoined-result)), Clause (2) is of the form

$$MB(G1, G2, [\wedge_{i=1}^{n} OCCURS(\alpha_i, G_{\alpha_i}, T_{\alpha_i}) \Leftrightarrow OCCURS(\gamma, G1 \& G2, T_E)]T\ 0)$$

where DURING(T_i, T_E)

and

$$\neg (\exists\ \alpha_j, \alpha_k\ ENABLE[(OCCURS(\alpha_j, G_{\alpha_j}, T_{\alpha_j})), (OCCURS(\alpha_k, G_{\alpha_k}, T_{\alpha_k}))])$$

Whereas the time intervals T_i must all be within the interval T_E, they may or may not overlap or be disjoint. In addition the conjoined actions cannot serve to enable one another.

Again, more briefly,

$$MB(G1, G2, GEN\ -Conjoined[(\alpha_i, \gamma, G1 \& G2, T_E)]T\ 0).$$

A discourse or dialogue for this variant is similar to that of the simultaneous action; the main difference is in exact times at which the actions are done.

5.3.3 Sequences of Actions

A somewhat more complicated variant of SharedPlan is one in which a sequence of actions together generate the desired action. For example, turning a door knob followed by pulling on the door knob together (under appropriate conditions, e.g., the door being unlocked) generate opening the door.

For SharedPlan3(G1, G2, Achieve(Sequence-result)), Clause (2) is of the form

$$MB(G1, G2, [\wedge_{i=1}^{n} OCCURS(\alpha_i, G_{\alpha_i}, T_{\alpha_i}) \Leftrightarrow OCCURS(\gamma, G1 \& G2, T_E)]T\ 0)$$

$$\text{where } START(T_1, T_E) \text{ and } FINISH(T_n, T_E) \text{ and } MEETS(T_i, T_{i+1})$$
$$\text{and } \neg (\exists\ \alpha_j, \alpha_k\ ENABLE[(OCCURS(\alpha_j, G_{\alpha_j}, T_{\alpha_j})), (OCCURS(\alpha_k, G_{\alpha_k}, T_{\alpha_k}))]).$$

Or, more briefly

$$MB(G1, G2, GEN\text{-}Sequence[\alpha_i, \gamma, G1 \& G2, T_E]T\ 0).$$

The interval T_i meets the next interval in the sequence and the intervals fully span the interval T_E of γ. The case of a sequence of actions achieving a desired action is not the same as an action enabling another action. Both α and β must be done to achieve γ, and α must be done before β, but α does not enable β. In the door knob example, turning the knob does not enable pulling on it; this can be seen quite simply by noting that one can also pull and then turn. The two actions together achieve opening the door.

5.3.4 Enablement

As Pollack [Pol86] has pointed out, the enabling relationship and the way it enters a plan introduces a number of complexities into the plan formalization and recognition process. Although, a detailed treatment of enabling relationships awaits further research, we can use a simple example to illustrate how enabling relations would fit within a SharedPlans.

Consider the utterance, "Please pass the butter." in the context of the speaker's eating dinner with the hearer, and the dinner including corn on the cob (and nothing else that is butter). Figure 2 shows the action decomposition relevant to this utterance and the buttering of a cob of corn. In place of the generation relation that is used in plan definitions for Pollack's SimplePlan and the SharedPlans presented in this paper, the plan sketched in this figure requires more complex action relationships. A portion of this decomposition will form the core of the beliefs of a SharedPlan that results in satisfying a condition on a private SimplePlan. The SharedEnablePlan(S, H, Achieve(Have-Butter)) satisfies the condition (Have-Butter) needed for S's SimplePlan of SimplePlan(...Achieve(buttered-corn)).

5.4 Evidence for SharedPlans

To refine our definitions of SharedPlans, we have begun collecting interaction records in the form of videotapes of agents performing tasks together.[7] Here we present results of our initial analysis of one interaction record. We videotaped two people in a natural setting building a piece of furniture (a swing glider); they undertook the task for reasons of their own (i.e., rather than due to a request from us). They were near each other at all times and

[7]In collecting our first interaction record we did not inform the agents that they were to collaborate.

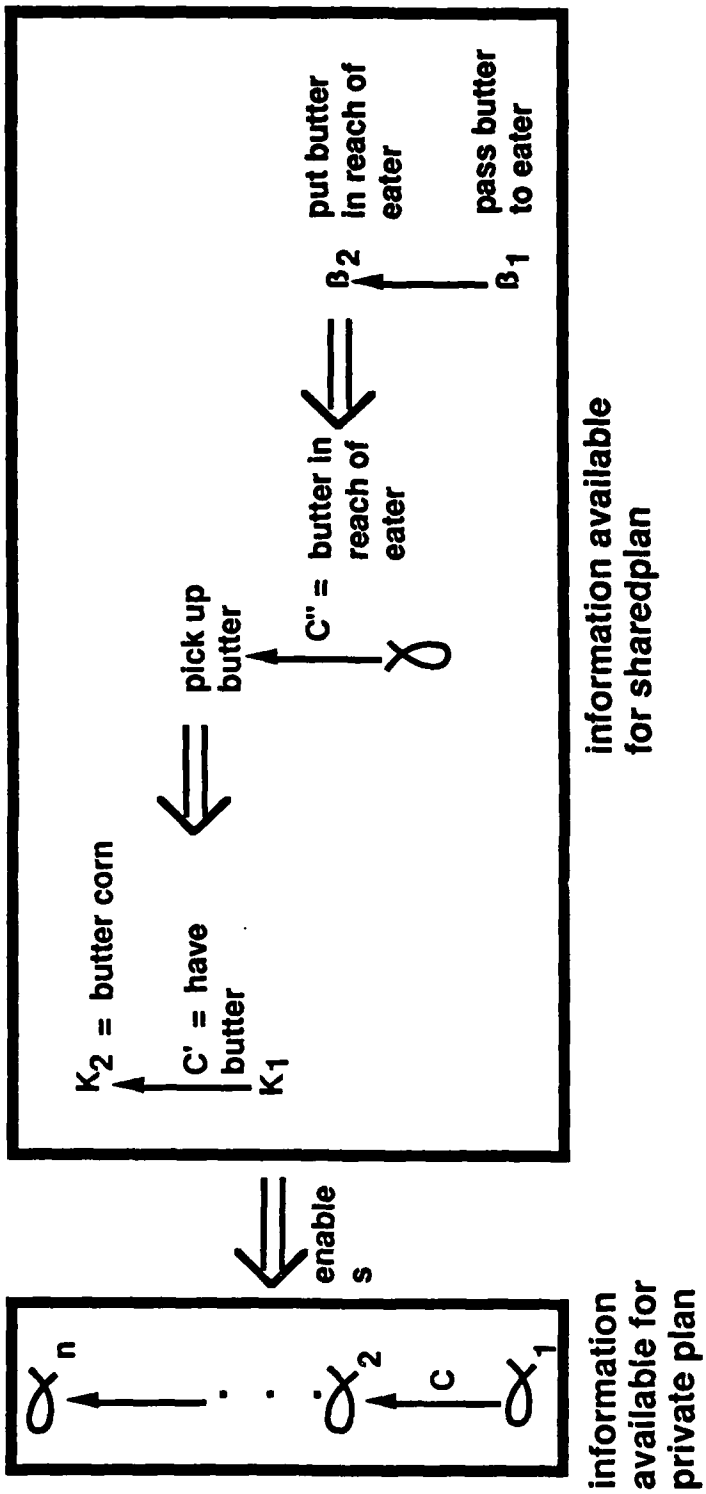


Figure 2: Private and SharedPlans for Passing Butter to Butter Corn

could see each other without obstruction. It should also be noted that the individuals knew each other quite well and were accustomed to working together (on scientific rather than physical tasks). We will discuss two portions of the tape that provide insight into the SharedPlan model because they indicate how agents R and L collaborated in the planning and acting cycle. Both portions include mistakes in the construction process, each serious enough for discussion. One mistake required R and L to disassemble a part of the partially assembled glider.

The first episode of planning concerns the construction of the base of the glider. The agents, having read all the directions that accompanied the unassembled furniture, begin with the first sub-task stipulated in the directions. Their task consists of an enablement of two actions; one action is a GEN-conjoined for each of them to place dowels in each end of the front and rear rails (the rails and the leg assemblies form a rectangular box-shaped structure). Each agent must individually also perform the two conjoined actions (in order to perform GEN-conjoined) of placing a dowel in one side of the front rail and one side of the rear rail; L will do the left side of the rails and R will do the right side. Their conjoined action enables a simultaneous action, namely attaching the rails to the leg assemblies. This action need not require simultaneity (one person could do it all alone), but perhaps since the agents have decided to work together, they decide to perform the task that way.

In this episode the agents discuss each part of the task (they do not discuss the whole base assembly) by announcing the action to be performed ("Place a small dowel in each end of the front and rear rails."). Some discussion and preparatory actions follow the announced

action.[8]\. The agents identify the parts to be used; they then discuss how to lay them out in the "right orientation," followed by doing so. Each time one agent offers a suggestion for what to do next (sometimes suggesting joint action with "let's ...", and sometimes suggesting action for the other individual as in "you might want to..."). A spoken response is not always forthcoming. When the other agent agrees, he may simply proceed with the action, thereby indicating assent by action that is visible to the other agent. The completion of the GEN-Conjoined action is not accompanied by any sort of channel checking or announcement of completion by both agents. R indicates completion by saying "then;" L does so by going onto the next part of the task.

The attachment of the leg assemblies is preceded by almost no discussion of the action. L simply announces "Oh then uh attach leg assemblies." Then L and R begin attaching the leg assemblies. Their first attempt meets partial failure after 7 seconds when R's front rail becomes detached (he mumbles "oops") and ends in complete failure after approximately 20 seconds. The full failure occurs when after attaching the pieces, L and R each attempt to tap on the structure; the whole thing collapses. In quick succession they each propose what is wrong with their simultaneous act and how to fix it: L offers "allright, we need to coordinate," while R offers "oh, actually here's a good thing. If you put the dowel in this end first then because it just slips on the other end, the dowel in this hole first and..." to which L responds simultaneously with the end of R's statement "right." They then proceed to carry out R's suggested course of action.

What can we learn from this brief portion of the interaction record? First, clearly agents

[8]We have not included a transcript of the videotape because we have not yet determined the proper way to make one that will indicate what was said, with proper pauses, overlapped speech and the like, what was done and where the agents looked. All of these behaviors are significant to the interaction record. Where we quote from the interaction record, we present only the words said and obvious utterance endpoints. During the workshop we will show these episodes from the tape.

make use of the cycle of planning (for whatever one takes planning to be), acting, and either restarting the cycle, or evaluating an unexpected outcome followed by (re)planning and acting. It is our belief on the basis of observing the entire session that R and L never have a plan that covers the entire assembly process. Rather they proceed in steps of planning a part of the assembly and doing it. They assume this cycle will produce the correct intermediate outcomes on the basis of the instructions and perhaps on the basis of beliefs that they will be able to repair any discrepancies.

We also learn that R and L do participate in a collaborative planning process; R and L decide what a sub-task is and sometimes discuss what one of them should do. There is sometimes explicit verbal assent and sometimes assent implied by undertaking action. Dissent, when it occurs, is handled rather indirectly. That is, the agents do not say "no," but instead state some proposition that implies their disagreement; they are very facile at picking up on each other's statements of this sort.

We also can observe that R and L make public, in a variety of ways, their beliefs and intentions. They often tell each other information about the assembly task and ask questions about something they believe the other will know. In addition the fact that they can see each other means they can rely on the other person recognizing visually their (intentions to do) actions that contribute to the desired outcome.

We can ask some specific questions about the nature of the failure that occurs in the first episode. On the basis of the SharedPlan model, we would expect that they could have erred in any one of three ways:

1. having false beliefs of each other's intentions to attach the rails to the leg

assemblies,

2. having false beliefs about their ability to execute the action required,

3. having false beliefs about time interval in which the action will begin [timing is critical in simultaneous action].

Judging from the interaction record, we have concluded that only beliefs two and three actually were held by one of the agents. L's comment about what is wrong indicates that he believes the error involved beliefs of the third type, while R's suggestion for what to do indicates an error in the way they are able to perform the overall enablement, i.e. errors in beliefs of the second type. His suggestion amounts to a different choice of actions to be performed, in part because the expectations he had about how tightly the parts would fit are not born out (and he tells L this during the episode).

False beliefs about the nature of relations among actions and the components used in performing an action play a significant role in the second episode of the interaction. That episode is quite extended, covering about 10 minutes of the 40 minute process. Early on in the interaction, R places washers on the 3 1/4" bolts that secure the front and rear rails to the leg assemblies. He does so out of the common belief that bolts require washers to keep the bolt head from digging into the wood (of the leg assemblies). This action is in error because it will later (several minutes later) leave the agents with too few washers for the rest of the assembly. The issue of where to put washers plagues the agents for the 10 minutes that follow.

That there may be a problem with using washers on the bolts occurs to L. Evidence of this comes from his interpretation of an ambiguously phrased question from R ("Does it show in the picture, is there supposed to be a washer?") L understands R to be asking about the

washers on the bolts. He immediately answers with a statement that is in fact false, though neither agent is aware of it at the time: "I'm sure there's a washer under this head." In fact R was asking about washers in use with nuts inside the rails. Eventually R deduces on the basis of the use of other washers that they have erroneously placed washers on the bolts. After more discussion, the agents agree that they must partially disassemble the structure in order to recover the washers that they need.

R and L's linguistic behavior indicates that they are very reluctant to tamper with their beliefs about actions once they have been undertaken. For example, when L decides that R is misplacing a nut (which he first calls a washer), his suggestion of this fact to R is quite tentative: "Um, < agent's name, > I'm I'm not sure the washer the the nut goes on the outside like that." A bit later L also asks about washers since he believes he sees an inconsistency in R's explanation about washers under the heads of the 2 1/4" bolts for the gliders straps[9] as compared to the washers under the heads of the 3 1/4" bolts for the rails. Their exchange indicates both the tentativeness of L's tampering and the conclusion that L finally draws and that R comes to hold as well.

L: How come we have a nut, a nut, a washer under here? Maybe we shouldn't have put washers there?

R: Where did we have washers?

L: Up here.

R: Oh I think it, well, it told us to. heh.

L: It showed washers on both ends? (said under his breath:) Or are we wrong?

R: I asked you about that. Oh no, we did that wrong. There was no washer under

[9]As an indication of their confusion, and to the confusion of anyone who views this tape, the agents refer to the bolts for the glider straps as nuts; sometimes they correct themselves and other times they do not.

that end of that nut.

While reluctant to tamper with beliefs about the relations among actions, R and L succeed in doing so. In the case above, the action of placing a nut under the rail bolts is believed to enable a condition of the proper state of the entire glider. While in fact the agents never resolve whether that state should be attained, they give up their belief in the enablement in favor of belief about necessary enabling states for other actions that have yet to occur (having washers for the bolts that attach the arm assemblies and back to the base). They do not at the time of giving up the first belief have a detailed belief about the upcoming enabling states. They believe there are some, but their beliefs are couched in terms of needing the washers in the future for some part of the assembly.

5.5 Concluding Comments

We have presented an initial formulation of SharedPlans, and definitions for a small set of actions involving collaborations among multiple agents. Analysis of exchanges in a collaborative endeavor of two people corroborates to some extent our stipulation of (mutual) beliefs in a plan, and the need for the construct of SharedPlan. However, much additional research remains To provide the basis for systems that can plan collaboratively with their users requires investigations in three areas: (1) definitions of the kinds of action relations that play central roles in collaborative activity; (2) an algorithmic description of collaborative planning; (3) the specification of the conditions under which collaboration will produce agreed upon common goals. We are focusing on the first of these three areas and in particular on the problem of specifying additional action relationships and the ways in which information about them is communicated among the participants in a collaborative planning process. The relations of causality and enablement (described briefly above) are

critical to plans, planning and acting [Geo87b]. Additional relationships enter into the complex (and still to be analyzed) activities that arise when actions cluster together to form a test and reaction to it. We plan to make use of the interaction record discussed in this paper to create definitions of those relations and then test our definitions against behavior in other videotapes.

References

[All79] J.F. Allen. *A Plan Based-approach to Speech Act Recognition.* Technical Report 131, University of Toronto, Toronto, Canada, 1979.

[All84] James Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123-144, 1984.

[Bru75] B.C. Bruce. *Belief systems and language understanding.* Technical Report 2973, Bolt Beranek and Newman Inc., Cambridge, Mass., 1975.

[Geo87a] Michael P. Georgeff. Planning. In Joseph Traub, editor, *Annual Review of Computer Science*, Annual Reviews, Inc., Palo Alto, Ca., 1987.

[Geo87b] M.P. Georgeff. Actions, processes and causality. In *Reasonings About Actions and Plans: Proceedings of the 1986 Workshop*, pages 99-122, Los Altos, CA, 1987.

[Gol70] A.I. Goldman. *A Theory of Human Action.* Princeton University Press, Princeton, NJ 1970.

[GS86] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175-204, 1986.

[GS88] B.J.Grosz and C. Sidner. Plans for discourse. In Cohen, Morgan, and Pollack, editors, *Intentions in Communication*, MIT Press, Cambridge, MA, 1988.

[KA86] H.A. Kautz and J.F. Allen. Generalized plan recognition. In *Proceedings*

of AAAI-86, the 5th National Conference on Artificial Intelligence, American Association of Artificial Intelligence, August 1986.

[Kon84] K. Konolige. *A Deduction Model of Belief and its Logics*. PhD thesis, Stanford University, 1984.

[Moo85] R.C. Moore. A formal theory of knowledge and action. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense Word*, pages 319-358, Ablex, Norwood, NJ, 1985.

[Mor86] L. Morgenstern. A first order theory of planning, knowledge and action. In J. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, pages 99-114, Los Angeles, CA, 1986.

[NCRF] Nils J. Nilsson, P.R. Cohen, S.J. Rosenschein, and K. Fertig. Intelligent communicating agents. Personal Communication.

[Pol86] Martha E. Pollack. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 207-214, Association for Computational Linguistics, New York, June 1986.

[Ros82] J.S. Rosenschein. Synchronization of multi-agent plans. In *Proc. Conf. Artif. Intell.*, pages 115-119, Stanford, CA, 1982.

[Sea88] J.R. Searle. Collective intentionality. In Cohen, Morgan, and Pollack, editors, *Intentions in Communication*, MIT Press, Cambridge, MA 1988.

[Sid83] C.L. Sidner. What the speaker means: the recognition of speakers' plans

in discourse. *International Journal of Computers and Mathematics*, 9(1):71-82, 1983.

[SSG79] D.F. Schmidt, N.S. Sridharan, and J.L. Goodson The plan recognition problem: an intersection of artificial intelligence and psychology. *Artificial Intelligence*, 10:45-83, 1979.

[Wil83] R. Wilensky. *Planning and Understanding*. Addison-Wesley, Reading, MA, 1983.

6. Domain Modelling for a Natural Language Processor[1]

Ralph M. Weischedel[2]

BBN Systems and Technologies Corporation

Abstract

In this paper, we discuss the following aspects of natural language understanding:

- The knowledge representation required
- Critical ontological decisions
- The multi-faceted role of a domain model
- A proposed high-level taxonomy that (we hope) is domain-independent and language-independent.

We have used a hybrid approach to representation, employing an intensional logic for the representation of the semantics of an utterance and a taxonomic language with formal semantics for specification of logical constants and axioms relating them. We employ a domain model which is a set of axioms expressed in the taxonomic knowledge representation system.

This combination has proved quite effective in BBN's natural language understanding and generation system (Janus). The paper lists how Janus employs the domain model and the limitations experienced in this approach.

[1]This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contracts N00014-85-C-0079 and N00014-85-C-0016.

[2]This brief report represents a total team effort. Significant contributions were made by Damaris Ayuso, Rusty Bobrow, Ira Haimowitz, Erhard Hinrichs, Thomas Reinhart, Remko Scha, David Stallard, and Cynthia Whipple. We also wish to acknowledge many discussions with William Mann and Norman Sondheimer in the early phases of the project.

6.1 Introduction

A *domain model*, for us, will be the specification of the logical constants of the domain and of axioms relating them. Though many [11,20] would argue that knowledge representation is critical to successful language processing, we have found that *a domain model is central to natural language processing*. In fact, all components of BBN's Janus natural language interface (including both the generation subsystem Spokesman and the understanding subsystem IRUS but not including the morphological analyzer), use the domain model in a central way.

In Janus, the meaning of an utterance is represented as an expression in an intensional logic. However, a logic merely prescribes the framework of semantics and of ontology. The *logical constants*, that is the constants (functions with no arguments), the other function symbols, and the predicate symbols, are abstractions without any detailed commitment to ontology nor to their semantics.

We have used NIKL as a basis for axiomatizing logical constants to explore whether a language with limited expressive power is adequate for core problems in natural language processing. Though we have found clear examples that argue for additional expressive power, they have been rare in our expert system and data base applications. Furthermore, the graphical editing, browsing, and maintenance facilities of KREME, a knowledge base editor, have made the task of porting Janus to new domains much easier.

Section 6.2 describes the formalisms chosen for domain modelling; Section 6.3 presents operators that derive new terms from existing logical constants; Section 6.4 briefly

describes the role the domain model plays in Janus; and Section 6.5 briefly summarizes two attempts at defining a general-purpose domain model.

6.2 Representation Commitments

To explore the virtue of some particular design decisions, we have made the commitments that follow.

- Intensional logic for representing the meaning of an utterance.

- The semantics of NIKL[5] to axiomatize logical constants, and therefore as the knowledge representation of the domain model.

They are decisions for the present; we neither assume that they will be the best for the foreseeable future, nor are religiously committed to them. Nevertheless, *as existing well-documented representation languages, they have enabled us to focus on formal domain specification rather than on issues in representation languages.*

In choosing NIKL to axiomatize the logical constants, we gain an inference algorithm, the classifier [18], which is incomplete, but which in practice has proven efficient. A further benefit in choosing NIKL is the availability of KREME, a sophisticated browsing, editing, and maintenance environment [1] that has proven effective in a number of projects having a taxonomic knowledge base. Though NIKL does not have the expressive power of first-order logic, we have found it adequate for the axioms needed by Janus in building a natural language interface to two military applications. This is discussed further in Section 6.5.

A quick review of the advantages of an intensional logic shows

- Both its structure and semantics can be close to that of natural language, leaving relatively little concern about its representational adequacy.

- Numerous linguistic studies provide semantic analyses that can be drawn upon.

However, its disadvantages include:

- The complexity of logical expressions is great even for relatively straightforward utterances using Montague grammar [14]. However, by adopting Montague's logic while rejecting Montague grammar, we have made several inroads into matching the complexity of the proposition to the complexity of the utterance; see [2].

- Realtime inference strategies to support definite reference is a challenge for so rich a logic. However, our hypothesis is that large classes of the linguistic examples requiring common sense reasoning can be handled using the limited inference algorithms of NIKL. Arguments supporting this hypothesis appear in [9, 15] for interpreting nominal compounds, in [3, 4, 19] for common sense reasoning about modifier attachment and in [20] for foregrounding phenomena in definite reference.

Nevertheless, the hybrid representation approach adopted has its limitations. Given time and world indices potentially on each logical constant, what is the meaning of typical relations such as those in Figure 1?

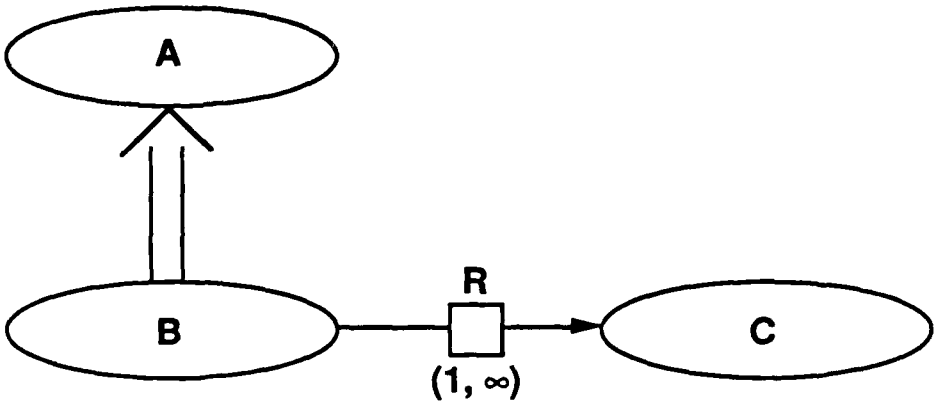


Figure 1: Two Typical Facts Stated in NIKL

In a first-order logic, the normal semantics would be

$$(\forall x)(B(x) \supset A(x))$$
$$(\forall x)(B(x) \supset (\exists y)(C(y) \wedge R(x, y))).$$

However, we need to interpret NIKL structures in light of time and world indices which can appear on each logical constants. Due to a suggestion by David Stallard, we interpret the structure in Figure 1 to mean

$$(\forall x)(\forall t)(\forall w)(B(x)(t, w) \supset A(x)(t, w))$$
$$(\forall x)(\forall t)(\forall w)(B(x)(t, w) \supset (\exists y)(c(y)(t, w) \wedge R(x, y)(t, w)))$$

Though this handles the overwhelming number of logical constants we need to axiomatize, we have found no way to capture (in NIKL) the semantics of predicates which should have intensions as arguments; therefore, they are unfortunately specified separately.

Examples that have arisen in our applications involve changes in a reading on a scale, e.g., *USS Stark downgraded from C1 to C4*. Approximately 5% of the vocabulary in our applications could not be handled with NIKL.

6.3 Some Global Ontological Decisions

Any concept name or role name in the network will be a logical constant. We use concepts only to represent sets of entities indexed by time and world. Roles are used only to represent sets of pairs of entities. Given that the axioms in our networks represent universal statements, several representational issues arise due to the nature of natural language semantics and of knowledge.

Suppose CATS and MICE are concepts in our network representing all cats and mice respectively. Following Scha [16], *the cats* will be represented as a term in the logic

(the X POWER (CATS)).

We will not enshrine that particular set of cats in the network, since only logical constants will appear in our networks.

Following Carlson's linguistic analysis [6,7] we will distinguish kinds, samples, and generics. Sometimes generic properties are associated with a kind, but not with every instance: as in *Cats are ferocious*. Here we assume there is a kind

(KIND CATS)

about which the property of ferociousness is asserted; the kind corresponding to CATS will not be in the network, since we do not choose to make the kind a logical constant. At times, an indefinitely specified set must be represented, as in *Mice were eaten.* An indefinitely specified set will be denoted by

(SAMPLE x MICE)

but the corresponding set will not be enshrined in the network.

Generic statements such as *Cats eat mice* are often encoded in a semantic network or frame system. However, representing the structure as in Figure 2 would not give the desired generic meaning, but rather would mean (ignoring time and world) that

$$(\forall x)(CATS(x) \supset (\exists y)(Mice(y) \wedge Eat(x, y))).$$

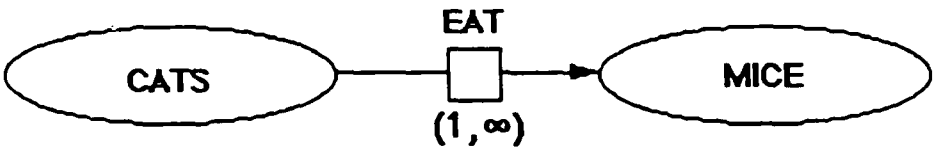


Figure 2: Illustration Distinguishing NIKL Networks from other Semantic Nets

Again, following Carlson's linguistic analysis, we would have a generic statement about the kind corresponding to cats, that these eat indefinitely specified sets of mice. Our formal representation (ignoring tense for simplicity) is

(GENERIC(LAMBDA(x)EAT(x,(SAMPLE y MICE)))) [KIND CATS].

An old notion proposed for definitions is the QUA link [10] for defining those entities that are exactly the fillers of a particular slot. USC/ISI [13] has proposed a similar mechanism for defining *pilot* as the filler of flying events. Nevertheless, QUA actually defines

$$(LAMBDA(x)(\exists\ y)(FLYING - EVENT(y) \wedge ACTOR(y, x))),$$

a predicate defining the set of items that have ever been the actor in a flight. Rather, one wants, we believe, a predicate defining the set of items that generically are the actors in a flight, i.e.,

$$(LAMBDA(x)(GENERIC(LAMBDA(x)(\exists\ y)(FLYING - EVENT(y) \wedge ACTOR(y, x))))).$$

6.4 Use of Domain Model in Janus

The domain model serves several purposes in Janus. First, in defining the logical constants of our semantic representation language, it provides the logical constants that lexical items map to. For instance, the lexical entry for *deploy* would state that it maps into an expression such as

(AND (DEPLOYMENT x) (DEPLOYMENT.LOCATION x y)),

where the two underlined items are a concept and role, respectively, in the domain model. The expression may be arbitrarily complex, allowing for anything from a one-to-one correspondence of logical constants and word senses to a complex decomposition of the word sense into primitives.

Second, the domain model provides the types (or sorts of a sorted logic) that form the primitives for selection restrictions. In the case of *deploy*, a MILITARY-UNIT can be the logical subject, and the object of a phrase marked by *to* must be a location.

Third, we define a mapping from each logical constant to the appropriate functionality in the data base, expert system, simulation program, etc. For a constant HARPOON-CAPABLE, that mapping would select those records of the unit-characteristics table with a "Y" in the HARP field.

The domain model has been used in Janus

- To interpret the meaning of *have* and *of* in context
- To filter possible modifier attachments while parsing
- To infer omitted prepositions when an input has a telegraphic style
- To infer lexical semantics from example utterances
- To organize, guide, and assist linguistic knowledge acquisition.

These topics are addressed elsewhere [2].

6.5 Experience with a General-Purpose Domain Model

One of the most serious impediments to widespread use of natural language processing technology is the effort (and therefore cost) involved in defining the knowledge needed by the natural language processor (NLP) to interpret/generate language for the given application. In Janus, since the domain model plays a central role, the domain model is most critical.

Were it possible to have a general purpose domain model which one extended and edited for a new domain/application, then the effort involved would be greatly cut. We investigated two approaches to building a general purpose domain model, which are covered in the following subsections.

6.5.1 Basic Vocabulary

In the *Longman Dictionary of Contemporary English* [12] (LDOCE) approximately 56,000 words are defined in terms of a base vocabulary of roughly 2,000 items.[3] We estimate that about 20,000 concepts and roles should be defined corresponding to the 2,000 multi-way ambiguous words.

The appeal, of course, is that if these basic notions were sufficient to define 56,000 words, they are generally applicable, providing a candidate for general purpose primitives.

[3]Though the authors try to stay within the base vocabulary, exceptions do arise such as diagrams and proper nouns, e.g., *Pennsylvania* and *Catholic Church*.

The course of action we followed was to build a KREME hierarchy for all of the definitions of approximately 200 items from the base vocabulary *using the definitions of those vocabulary items themselves in the dictionary*. In this attempt, we encountered the following difficulties:

- Definitions of the base vocabulary often involved circularity.

- Definitions included assertional information and/or knowledge appropriate in defeasible reasoning, which are not fully supported by NIKL. The first definition of *cat* is "a small four-legged animal with soft fur and sharp claws, often kept as a pet or for catching mice or rats."

- Multiple views and/or vague definitions and usage arose in LDOCE. For instance, the second definition of *cat* (p. 150) is "an animal *related to this* such as the lion or tiger" (italics added). Such a vague definition helped us little in axiomatizing the notion.

It was thus clear that much careful thought would be needed to axiomatize by hand the LDOCE base vocabulary if general-purpose primitives were to result. Consequently, it seemed most appropriate to define general abstractions which would set an ontological style, then to use the more concrete primitives of LDOCE within the style set by the hand-crafted abstractions. A more detailed analysis of our experience is presented in Chapter 7.

6.5.3 Abstractions

An alternative course of action is to focus on the highest level of abstractions since they provide a style in which more concrete, general-purpose notions fit. In this sense, it has goals similar to the USC/ISI "upper structure" [13], which seems tied to systemic linguistics rather than to a more general ontological style.

All of our concept abstractions divide into either a PROCESS, i.e., that which changes things, or an OBJECT, that which can change, or a DESCRIPTION, which is a property. A PROCESS normally has roles associated since we reify processes. Though reifying processes could be attributed to the limitations of the formalism, Davidson [5] has argued for reification on semantic grounds. Some roles are thematic, analogous to notions in [8, 17], e.g., AGENT, PATIENT, BENEFICIARY, INSTRUMENT; and some are not linguistically motivated, e.g., STATE.BEFORE, and STATE.AFTER. Actions therefore come under the PROCESS hierarchy, as do events (actions with a specific time of occurrence). Unfortunately, it is impossible in this short note to go through all of the 45 descendants of PROCESS, nor all 50 descendants of DESCRIPTION, nor all of the 38 roles, nor all 50 descendants of OBJECT, created thus far.

6.6 Conclusions

Our conclusions regarding the hybrid representation approach of intensional logic plus NIKL-based axioms to define logical constants are based on three kinds of efforts:

- Bringing Janus up on two large expert system and data base applications within DARPA's Fleet Command Center Battle Management Program. The combined lexicon in the effort is over 6,000 words (not counting morphological variations).

- The efforts synopsized here towards a general purpose domain model.

- Experience in developing a set of acquisition tools integrated with the domain model acquisition and maintenance facility (KREME).

First, *a taxonomic language* with a formal semantics can supplement a higher order logic in support of efficient, limited inferences needed in a natural language processor. Based on our experience and that of others, the axioms and limited inference algorithms can be used for classes of anaphora resolution, interpretation of *have* and *of*, finding omitted relations in novel nominal compounds, and selecting modifier attachment based on selection restrictions.

Second, *an intensional logic can supplement a taxonomic language in trying to define word senses formally*. Our effort with LDOCE definitions showed how little support is provided for defining word senses in a taxonomic language. A positive contribution of intensional logic is the ability to distinguish universal statements from generic ones from existential ones, definite sets from unspecified ones, and necessary and sufficient information from assertional information.

Third, *the hybridization of axioms for taxonomic knowledge with an intensional logic does not allow us to represent all that we would like to, but does provide a very effective*

engineering approach. Out of several thousand lexical entries, only a handful represented concepts inappropriate for the formal semantics of NIKL.

Fourth, we hypothesize that *a large general purpose domain model is feasible and highly desirable.* Our experience has not proved that hypothesis, for the work is still young.

There are aspects that could properly be termed *domain modelling* which we have not addressed either in this paper or in Janus. Procedural knowledge about events sequences, goals, and plans is an area of research which we do not believe to be adequately represented via NIKL.

REFERENCES

[1] Abrett, G. and Burstein, M. The KREME knowledge editing environment. *Int. J. Man-Machine Studies* 27:103-126, 1987.

[2] Ayuso, D.M., Weischedel, R.M., Bobrow, R.J. Semantic Interpretation in the Janus Understanding and Generation System. 1988.

[3] Bobrow, R. and Webber, B. PSI-KLONE: Parsing and Semantic Interpretation in the BBN Natural Language Understanding System. In *Proceedings of the 1980 Conference of the Canadian Society for Computational Studies of Intelligence.* CSCSI/SCEIO, May, 1980.

[4] Bobrow, R. and Webber, B. Knowledge Representation for Syntactic/Semantic Processing. In *Proceedings of the National Conference on Artificial Intelligence.* AAAI, August, 1980.

[5] Brachman, R.J. and Schmolze, J.G. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 9(2), April, 1985.

[6] Carlson, Gregory. *Reference to Kinds in English.* PhD thesis, University of Massachusetts, 1977.

[7] Carlson, G. *Reference to Kinds in English.* Garland Press, New York, 1979.

[8] Fillmore, Charles J. The Case for Case. In Bach, Emmon, & Harms, Robert T. (editors), *Universals in Linguistic Theory*, chapter 1, pages 1-88. Holt, Rhinehart, & Winston, New York, NY, 1968.

[9] Finin, T.W. The Semantic Interpretation of Nominal Compounds. In *Proceedings of The First Annual National Conference on Artificial Intelligence*, pages 310-312. The American Association for Artificial Intelligence, The American Association for Artificial Intelligence, August, 1980.

[10] Freeman, M. The QUA Link. In Schmolze, J.G. and Brachman R.J. (editors), *Proceedings of the 1981 KL-ONE Workshop*, pages 55-65. Bolt Beranek and Newman Inc., 1982.

[11] Hobbs, J.R. and Martin, P. Local Pragmatics. In John McDermott (editor), *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 520-523. International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, Inc., August, 1987.

[12] *Longman Dictionary of Contemporary English.* Essex, England, 1987.

[13] Mann, W.C., Arens, Y., Matthiessen, C., Naberschnig, S., and Sondheimer, N.K. *Janus Abstraction Structure -- Draft 2.* Technical Report, USC/Information Sciences Institute, 1985.

[14] Montague, Richard. The Proper Treatment of Quantification in Ordinary English. In J. Hintikka, J. Moravcsik and P. Suppes (editors), *Approaches to Natural Language*, pages 221-242. Reidel, Dordrecht, 1973.

[15] Planes, D. Ayuso. The Logical Interpretation of Noun Compounds. Master's thesis, Massachusetts Institute of Technology, June, 1985.

[16] Scha, R. and Stallard, D. Multi-level Plurals and Distributivity. In the *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 17-24. Association for Computational Linguistics, June, 1988.

[17] Schank, R.C. The conceptual analysis of natural language. In Randall Rustin (editor), *Natural Language Processing*, pages 291-309. Algorithmics Press, New York, 1973.

[18] Schmolze, J.G., Lipkis, T.A. Classification in the KL-ONE Knowledge Representation System. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. 1983.

[19] Sondheimer, N.K., Weischedel, R.M., and Bobrow, R.J. Semantic Interpretation Using KL-ONE. In *Proceedings of COLING-84 and the 22nd Annual Meeting of the Association for Computational Linguistics*, pages 101-107. Association for Computational Linguistics, Stanford, CA, July, 1984.

[20] Weischedel, R.M. Knowledge Representation and Natural Language Processing. *Proceedings of the IEEE* 74(7):905-920, July, 1986.

7. Summary of Conclusions from the Longman's Taxonomy Experiment

T. Reinhardt C. Whipple
BBN Systems and Technologies Corporation

7.1 Overview of Experiment

As part of the continuing research into knowledge representation schemes for natural language processing, a subset of primitives was chosen from the approximately two-thousand primitive concepts listed in the back of the Longman Dictionary of Contemporary English [3]. The hope was that by using existing knowledge representation tools, specifically KREME [1, 2], a domain-independent model which would provide a core set of concepts that might be used by multiple domain-specific applications.

7.2 Method

Given the size and complexity of the domain model, we attempted a top-down approach, one that sought to focus upon the *organizational* as opposed to the *definitional* aspects of the task. Initially, five primitives were chosen: object, time, space, idea and state. Since each had numerous definitions associated with it, the symbolic names were suffixed with tags indicating the part of speech and definition

used. For instance, **Thing[n1]**, indicated that the first definition of **Thing** as a noun was chosen.

KREME provides a general facility for constructing and maintaining taxonomies. Specifically, KREME is a flexible "knowledge representation, editing and modelling environment" that

- Enables designers to define **Concepts** and **Roles**. Concepts are unary relations that represent organizational or definitional entities. **Roles** are restricted binary relations that map *domain* concepts to *range* concepts;

- Organizes binary and unary relations into *subsumption hierarchies*, i.e., partial orderings induced by a *subsumption* operator; and,

- Provides an interactive editing facility for creating, deleting and modifying nodes within these networks.

Concepts represent definitions, in the case of actual Longman's primitives, or organizational markers, in the case of "synthetic" nodes. **unit[n2]** denotes the second noun definition of **unit** — *a quantity or amount used as a standard of measurement.* Concepts without "[ ]"'s suffix denote synthetic concepts that were added to organize and root sublattices: **primitive.unit**, for instance, is the synthetic parent node for all metric and avoirdupois units of measurement. Roles were used to describe the functional characteristics of a node. For instance, **unit.of** is a role whose domain is

thing and whose range is **unit[n2]**. For example, a physical object may have a **weight** expressed in **pounds** — which is a subclass of **unit[n2]**.

As definitions were added, they were examined for their primitiveness. If a definition used terms that were Longman's primitives, they were assimilated as well. For example, Longmans defines **Thing[n4]** as *a subject; matter. Subject*, it turns out, is a primitive comprised of seven definitions and *matter* nine. Two definitions of *subject* appeared germane:

Subject[n1] [(of)] the thing that is dealt with or represented in a piece of writing, work of art, etc., and,

Subject[n2] something being talked about or considered.

And two definitions for *matter* appeared to apply:

Matter[n1] a subject to which one gives attention; situation or affair, and,

Matter[n4] a subject itself as opposed to the form in which it is spoken or written about.

In this example, note that an indirect circularity arises in using the term "subject" in the definition of **Matter** since both "subject" and "matter" are expressed as terms in the definition of **Thing[n4]**. In general, such definitional circularity was unavoidable since a majority of the Longman definitions were circular. No attempt was made to

"flatten" out such definitions since it was the intent of the project to faithfully rely upon the Longman taxonomy.

Nevertheless, some effort was made to keep the resulting taxonomy small, but representative. Hence, the recursive inclusion of terms used in definitions was kept to a manageable minimum. The current taxonomy comprises only two hundred and twenty concepts, i.e., unary relations, and thirty roles, i.e., restricted binary relations. In addition to the Longman primitives, "synthetic" nodes were added to enhance readability and provide some structure to the lattice.

In one instance we attempted to extend the sublattice rooted at **unit[n2]**, *amount or quantity used as a standard of measurement*, to leaf nodes. In this case, we took the taxonomy out to specific units of measurement in both the metric and avoirdupois systems.

7.3 Results

Our results bear out the adage that "the more specific the taxonomy, the more useful and accessible its descriptive power." The more specific a knowledge representation model is, the more likely that it will provide a sufficiently directed core of concepts for a given application. Longman's selection of primitives includes a potentially viable subset of definitions which will provide the foundation for such a model.

7.3.1 The Limits of Logic-Based Representation

KREME is organized around a notion of classification in the sense that a unary or binary relation may be uniquely classified in a lattice of other such relations.[1] Such classification is a powerful tool in domains that are amenable to partitioning by some partial ordering operator — a *subsumes* relation, for example.

In addition, KREME classification provides an elegant foundation upon which to introduce higher level functionalities — such as disjunction and equivalence operators and inverse relations. These wield "big descriptive sticks" so long as some underlying assumptions remain true: That the universe of discourse is finite and enumerable; that all relations between entities are bounded and binary; and, that the elements of discourse are *definitional* entities as opposed to *assertions*.

In the current experiment, though, classification brought little computational or descriptive leverage to bear on the problem of organizing such general knowledge — other than providing a formal notion of hierarchy. Disjointness operations, for instance, proved (at least thus far) difficult to maintain until we approached leaf nodes. Consider the following example, whose complete sublattice is illustrated in Figure 1, where disjointness was applied.

---

[1] A semi-lattice, actually, since not all nodes are defaultly terminated by an encompassing "most specific node."

Frequently, Longmans defined more complex units of measurement in terms of their relation to more primitive "leaf nodes:" **Rate[n1]** is defined as *a quantity such as value, cost or speed, measured by its relation to some other amount*, which is a definition in terms of terminal concepts that lacks a certain functional capacity; it does not describe the computation necessary to compute a rate as measured in compound units such as **unit[n2] per unit.of.time**.

In earlier formulations (see Figure 1), **units.of.time** were subsumed by **metric.unit** and **avoirdupois[n1]** in an attempt to capture the fact that both systems of measurement incorporated **units.of.time** in calculations of rate, etc. This led to a problem where **metric.unit** and **avoirdupois[n1]** could no longer be maintained as conceptually disjoint.

A cleaner representation resulted from moving **units.of.time** to the same level as **avoirdupois[n1]** and **metric.units**, and declaring them all disjoint, see Figure 2. But, we still needed to account for "compound descriptions," i.e., units of measurement that are composed of two or more primitive units, such as "miles per hour." The synthetic nodes, **compound.unit** and **primitive.unit**, were required in order to represent concepts such as "miles per hour" which is a functional relationship between two disjoint concepts.

In summary, disjunction assertions were attempted at various points in the lattice. In the case of disjointnesses on Longman's primitive concepts, though, most were abandoned as sibling nodes were added because of unforseen relations between pairwise siblings, as in the case illustrated in Figure 1. A possible explanation for this might be that all concepts converge in the limit; hence disjointness is at or very near the

level of terminals, i.e., leaves. Finally, disjointness seemed easier to maintain on synthetic nodes — but this is to be expected since these nodes were relatively linear (i.e., lacked the circularity of Longman's definitions) and were created to fill functional voids and provide descriptive structure to the overall lattice.

7.3.2 Synonyms, Aliases

Longman's primitives include terms that are synonyms or aliases. An instance that was left unexplored was the inclusion of the Imperial system. In that case, either some labelling scheme might have been employed to textually distinguish imperial from avoirdupois units of the same name or the KREME synonym facility would have had to have been augmented to provide the desired functionality.

KREME, as it presently stands, treats synonyms as non-primitive specializations of a concept. For instance, to define node C' as a synonym for C is to say that C' is a non-primitive specialization of C.

Although largely untested, it appears that this scheme is plausible. Unfortunately, it does not address the question of *aliases*, such as Imperial versus metric or avoirdupois **ton**. Of course, another interpretation is that the defintion could be sufficiently "gutted" to *a measure of weight* — see Figure 2, as in the the case of **ton[n1]**. In this case it's encumbent upon the process driving the taxonomy to establish the context of the reply.

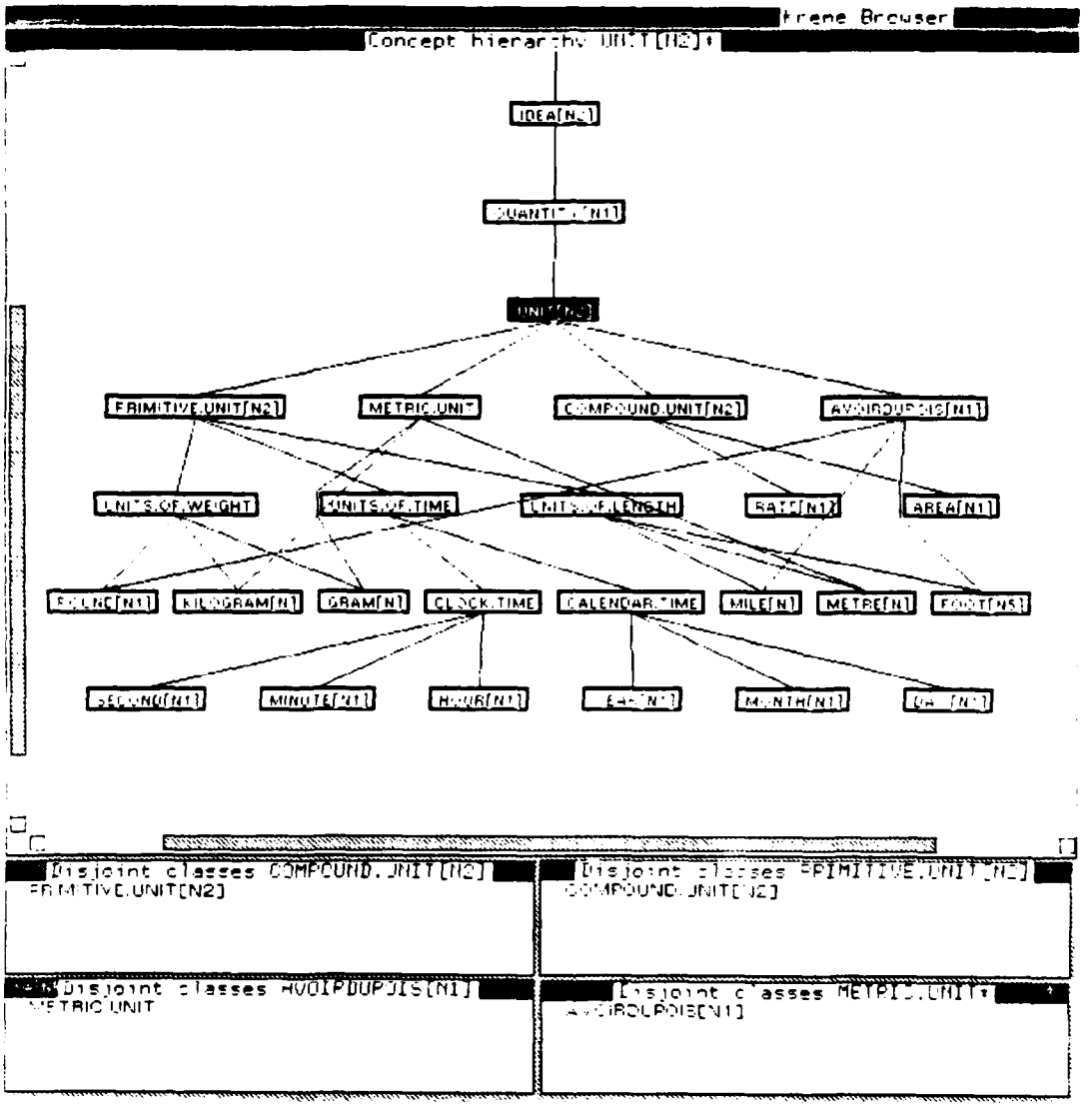


Figure 1: Sublattice depicting the relationship between units of measurement in both the avoirdupois and metric systems. Note that compound and primitive units of measurement are disjoint, but assertions about the disjointness of **metric.unit** and **avoirdupois[n1]** had to be retracted at **units.of.time**.

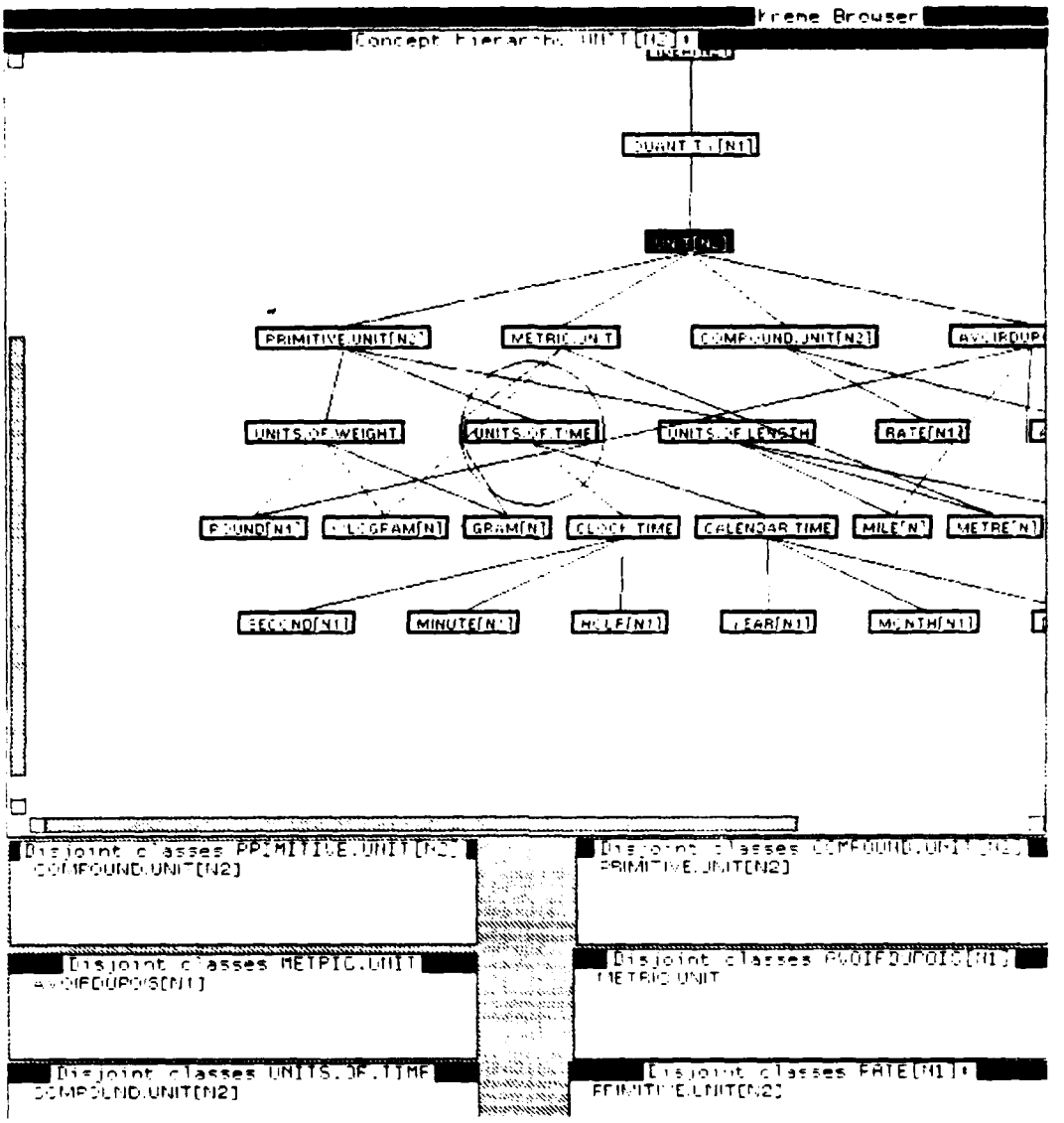


Figure 2: Edited sublattice from Figure 1, above. Note that **compound.unit** and **primitive.unit**, as well as **avoirdupois[n1]**, **metric.unit** and **units.of.time** are now asserted to be disjoint.

7.3.3 Definitional Weaknesses

A major problem with trying to strictly adhere to Longman's set of primitives is the definitional weaknesses of the primitives. Consider the definition of **matter[n3]** as *the physical material of which everything that we can see or touch is made, as opposed to thought or mind; solids, liquids and gases.* Note that this definition is *weak* in the sense that it is bifurcated, i.e., it consists of two definitions: *the physical material of which everything that we can see or touch is made, as opposed to thought or mind*; and *solids, liquids or gases.* Such definitions appear repeatedly throughout the dictionary. It was felt that *stronger* definitions comprised of singular statements would have gone far in alleviating confusion and ambiguity.

7.3.4 Using Roles More Effectively

The current taxonomy is "concept heavy" and might be improved by more effective use of roles, i.e., restricted binary relations, especially in determining usage. Consider, for instance, the definition of **State[n1]** as *(of) a condition in which a person or thing is; a way of being, feeling, or thinking considered with regard to its most important or noticable quality.* Attached to the concept, **State[n1]**, might be the role, **state.of**, which maps from its domain, (∨ **person[n1]**; **thing**) to the range, **condition[n1]** — *a state of being or existence.*

A potential problem with this approach is the lack of disjunction operators in KREME. Assuming that the classifier could represent disjunction as a synthetic node whose generalizations are the disjoins, we are still faced with the problem of exhaustive

search over an inflated network. In the case of the "state of the state," for instance, this might prove expensive.

Similiar problems arise with the transitive verb, **to state[v1]** — *to say, express, or put into words, esp. formally.* Here, the problem is worse since we have an ill-specified domain and a range that is the union of written and spoken words. Notwithstanding these problems, is the issue of how to capture the notion of *esp. formally* as it occurs in the definition.

7.3.5 Idiosyncratic Usage

That Longmans' is an *English* and not *American* dictionary is important. Clearly numerous idiosyncracies arise, consider the definition **of meal**, for instance: *an amount of food eaten at one time, usu. consisting of two or more dishes.* Consider now its derogatory usage: **make a meal** of *to give (something) more effort, consideration, or time than it deserves.* And, of course, there are countless others.

Besides word differences, of course, usage must be taken into account. For instance, one can *take a bath* only in American English whereas one will *have a bath* in British English.

7.3.6 Heterarchies vs. Hierarchies

All classification schemes display a degree of *arbitrariness*. The relationship between two nodes in one hierarchy might be completely inappropriate in another hierarchy. In reality, hierarchies are secondary revisions, attempted explanations in retrospect, of essentially *heterarchical* structures of human perception. How (and it's an open question as to whether) human beings classify depends upon their current *viewpoint* or *perspective*. Consider that a botonist might classify plants differently than a gardener, although both are speaking about the same entity.

Should the current experiment be extended to attempt to include a *complete* (in any sense) collection of primitives, or to handle colloquial or idiosyncratic usage, the problem of the essentially heterarchical nature of human experience might have to be taken into account.

7.3.7 Less Complex Interrelations

Pushing aside the issue of heterarchies, and momentarily assuming that common sense and practical linguistic knowledge could be represented hierarchically, we're still mired in the semantics of *links* in such networks. In other words, we'll assume that one could meaningfully construct concepts as unary logical entities and maintain that their relationships are such that they form a partial ordering, we are still stuck with the problem of the semantics of a link: what does it mean to say that C_i is above C_j in a particular network? Clearly the ISA semantics of Kreme and the KL family of languages is insufficient. Often we need to express "part whole," "one to many" and

"many to one" relationships. KREME semantics lacks the facility for expressing such relationships directly.

7.3.8 Definitional vs. Assertional Knowledge

It's likely an open question as to how much of our knowledge of the world is strictly definitional and therefore reducible to a classification problem and how much is assertional which certainly isn't classified in the same sense.

That KREME doesn't distinguish between "definitional" entities and assertions only exacerbates this problem. Consider **person[n1]** — *a human being considered as having a character of his or her own, or as being different from all others.* Clearly the first part of the definition, *a human being [. . .]* is definitional where the remaining definition is problematic. A potential solution is the ability to create *individual* instances of a particular class. In this case, perhaps, such assertional knowledge can be used in an active capacity, e.g., to ensure that two individuals are in fact different. In the present experiment, however, this avenue remained unexplored.

7.4 Final Commentary

One reason that Longman's works as a dictionary of English for non-English speaking people is that its audience is familiar with *some natural language.* They might not speak English but they speak some natural language and not LISP, for example.

Human languages exploit the kinds of ambiguities that classification systems, at least classification systems based upon formal logic, seek to avoid. Circular definitions, for instance, can be correctly interpreted by humans and the problem of strict linear interpretations do not arise as they do in our formal hierarchy.

The most promising subset of primitives are the more concrete, terminal concepts, i.e., those concepts most frequently used to describe the world at large. Re-building thc hierarchy from the bottom up with the freedom to create synthetic concepts not restricted to Longman's primitives might result in a more descriptively adequate and perspicuous model. The feeling persists, however, that a general domain-independent model built on Longman's selection of primitives is at once too general and too specific. More promising might be to focus on modelling more restricted taxonomies of specific branches of knowledge, for instance a domain-independent biological domain model.

References

[1] Abrett, G., Burstein, M., Gunshenan, J., and Polanyi, L. *KREME: A User's Introduction.* Technical Report 6508, Bolt Beranek and Newman Inc., 1987.

[2] Abrett, G. and Burstein, M. *The BBN Knowledge Acquisition Project: Phase 1 — Final Report; Functional Description; Test Plan.* Technical Report 6543, Bolt Beranek and Newman Inc., 1987.

[3] *Longman Dictionary of Contemporary English.* Longman Group U.K. Limited, Longman House, Burnt Mill, Harlow, Essex, CM20 2JE, England, 1987.

8. Presentations and Publications

8.1 List of Presentations

B. Goodman, "Communication and Miscommunication," Harvard University, Cambridge, Ma., April 1985.

B. Goodman, "Repairing Reference Identification Failures by Relaxation," Association of Computational Linguistics, Annual Meeting, Chicago, Il., July 1985.

B. Goodman., "Micommunication and Plan Recognition," User Modelling Workshop, Maria Laach, West Germany, August 1986.

B. Goodman, "Reference and Reference Failure," Theoretical Issues in Natural Language Processing III (TINLAP3), New Mexico State University, Las Cruces, New Mexico, January 1987.

B. Goodman and D. Litman, "Aiding Design with Constructive Plan Recognition," AAAI Workshop on Plan Recognition, St. Paul, Mn., August 1988.

A. Haas, "The Case for Domain-Specific Frame Axioms," Workshop on Logical Approaches to the Frame Problem, University of Kansas, University of Kansas, April 1987.

E. Hinrichs, "A Compositional Semantics for NP Reference and Aktionsarten," West Coast Conference on Formal Linguistics, March 1986.

E. Hinrichs and L. Polanyi, "Pointing the Way: A Unified Treatment of Referential Gesture in Interactive Discourse," at the 22nd Annual Meeting of the Chicago Linguistic Society, April 17-19, 1986.

E. Hinrichs, "A Compositional Semantics for Directional Modifiers in English - Locative Case Reopened," at the 11th International Conference on Computational Linguistics, University of Bonn, August 25-29, 1986.

E. Hinrichs, "Pointing, Language and the Visual World: Towards Multimodal Input and Output for Natural Language Dialog Systems," Panel on Pointing, Language and the Visual World, 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 1987.

L. Polanyi, "The Dynamic Discourse Model," Linguistics Department, UCLA, Los Angeles, Ca., March 1985.

L. Polanyi, "Discourse Analysis with the DDM," Psychology Department, Stanford University, Stanford, Ca., April 1985.

L. Polanyi, "A Theory of Discourse Structure and Discourse Coherence," Chicago Linguistics Society, 21st Annual Meeting, April 1985.

L. Polanyi, "Modelling Natural Language Discourse," Cognitive Science Seminar, SUNY Buffalo, Buffalo, N.Y., May 1985.

L. Polanyi, "Discontinuous Constituents in Discourse," Conference on Discontinuous Constituents, University of Chicago, Chicago, Il., July 1985.

L. Polanyi and R. Scha, "The Dynamic Discourse Model: A Formal Approach to Discourse Segmentation," Association for Computational Linguistics, Annual Meeting, Chicago, July 1985.

L. Polanyi, "Modelling the Linguistic Structure of Discourse," Invited Workshop, Linguistics Institute of the Linguistics Society of America, Georgetown University, Washington, D.C., July 1985.

L. Polanyi, "Discourse Analysis from a Linguistic Point of View," Boston Interaction Research Group, January 1986.

L. Polanyi, "A Linguistic Approach to Discourse Analysis," Massachusetts Interdisciplinary Discourse Analysis Seminar, February 1986.

L. Polanyi, "A Formal Model of Discourse Structure," 2nd Cognitive Science Seminar, Tel Aviv University, Tel Aviv, Israel, April 1986.

L. Polanyi, "Discourse Syntax, Discourse Semantics, Discourse Semiotics: The Case of the Discourse Pivot," Cognitive Science Series, University of Buffalo, Buffalo, N.Y., April 1986.

L. Polanyi, "Narrative Organization and Disorganization, Workshop Symposium on the Acquisition of Temporal Structures in Discourse, University of Chicago, Chicago, Il., April 1986.

L. Polanyi, "Pointing and Language," Workshop on Integration of Natural Language and Non-Verbal Information, Milan, Italy, August 1987.

R. Scha, "The Role of Intonation in Marking Discourse Structure," IEEE Workshop of Speech Recognition, Harriman, N.Y., December 1985.

J. Schmolze, "On Representing Some Everyday Phenomena for Planning Purposes," University of Massachusetts, Amherst, Ma., May 1985.

J. Schmolze, "Physics for Robots," Brandeis University, Waltham, Ma., March 1986.

J. Schmolze, "Semantics for NIKL," MIT Workshop on Terminalogical Languages, MIT, Cambridge, Ma., July 1986.

J. Schmolze, "Physics for Robots," Conference of the American Association for Artificial Intelligence, Philadelphia, Pa., August 1986.

C. Sidner, "Issues in Pragmatics: Plan Recogntion and Discourse Theory," TANLU Workshop, May 1985.

C. Sidner, "Discourse Structure and the Proper Treatment of Interruptions," International Joint Conference on Artificial Intelligence, Los Angeles, Ca., August 1985.

C. Sidner, "AI, Computational Linguistics and Discourse Theory," Massachusetts Interdisciplinary Discourse Analysis Seminar, March 1986.

C. Sidner, "Modelling Discourse Structure: The Role of Purpose in Discourse," Sixth Annual Canadian AI Conference, Montreal, Canada, May 1986.

C. Sidner (with B. Grosz), "Plans in Discourse," SDF Benchmark Series in Computational Linguistics III, Plans and Intentions in Communication and Discourse, Monterey, Ca., March 1987.

C. Sidner, "Plans in Discourse," BBN Labs, Cambridge, Ma., April 1987.

C. Sidner, "The Discourse Structure Theory," Linguistics Institute Seminar on Discourse, Stanford University, Stanford, Ca., July 1987.

C. Sidner and B. Grosz, "Plans for Discourse," AI Seminars Series, BBN Laboratories Inc., Cambridge, Ma., January, 1988.

N. S. Sridharan, Panel on User Modeling, Ninth International Joint Conference on Artifical Intelligence, Los Angeles, Ca., August 1985.

N. S. Sridharan, Panel on the Role of AI in Design, 1985 International Conference on Computer Design, Rye, N.Y., October 1985.

N. S. Sridharan, Workshop on Distributed Artificial Intelligence, Sea Ranch, Ca., December 1985.

N. S. Sridharan, Computer Science Colloquium, University of Pennsylvania, Philadelphia, Pa., January 1986.

N. S. Sridharan, Computer Science Colloquium, Northeastern University, Boston, Ma., February 1986.

N. S. Sridharan, "Semi-Applicative Programming," BBN AI Seminar Series, January 1986.

N. S. Sridharan, Workshop on Future Directions in Computing, sponsored by the Army Research Office, Seabrook Island, S.C., May 1986.

N. S. Sridharan, Workshop on Artificial Intelligence, sponsored by Indian Ministry of Defense, Bangalore, India, June 1986.

M. Vilain, "The Restricted Language Architecture of a Hybrid Representation System," Ninth International Joint Conference on Artifical Intelligence, Los Angeles, Ca., August 1985.

M. Vilain, "KL-Two and Hybrid Knowledge Representation, University of Rochester, Rochester, N.Y., November 1985.

M. Vilain, "Abstraction and Classification in a Hybrid Representation System, MIT A.I. Laboratory, MIT, Cambridge, Ma., March 1986.

M. Vilain (with H. Kautz), "Constraint Propagation Algorithms for Temporal Reasoning," Conference of the American Association for Artificial Intelligence, Philadelphia, Pa., August 1986.

M. Vilain, "Recent and Forthcoming Developments in KL-Two," MIT Workshop on Terminological Languages, MIT, Cambridge, Ma., July 1986.

M. Vilain, "Medium Grain Parallelism for Knowledge Representation," Brown University, Providence, Rhode Island, May 1987.

M. Vilain, "Parallel Truth Maintenance Techniques." Presentation given at the DARPA Natural Language Workshop, SRI International, November 11, 1987. Also presented at XEROX Palo Alto Research Center and at BBN Labs.

M. Vilain, "A Parallel Truth Maintenance System," AAAI Workshop on Parallelism in Machine Intelligence, St. Paul, Mn., August 1988.

8.2 List of Publications

B. Goodman, *Communication and Miscommunication*, BBN Laboratories Inc., Cambridge, Ma., Technical Report No. 5681, October 1985.

B. Goodman, "Repairing Reference Identification Failures by Relaxation," *Proceedings of the 23rd Annual Meeting of the ACL*, July 1985.

B. Goodman, "Reference and Reference Identification Failures," *Computational Linguistics*, Vol. 12, No. 4, 1986 (a revised version also appears as *Rule-Based Relaxation of Reference Identification Failures*, Technical Report No. 396, Center for the Study of Reading, University of Illinois at Urbana-Champaign, 1986).

B. Goodman, "Reference and Reference Failures," in *Proceedings of Theoretical Issues in Natural Language Processing-3*, TINLAP-3, New Mexico State University, Las Cruces, New Mexico, 1987 (also appears as Technical Report No. 398, Center for the Study of Reading, University of Illinois at Urbana-Champaign, 1986).

B. Goodman, "Repairing Reference Identification Failures by Relaxation," in *Communication Failure in Dialogue and Discourse*, Ronan Reilly (ed), North-Holland, Amsterdam, 1987.

B. Goodman et. al., *Research in Knowledge Representation for Natural Language Communication and Planning Assistance: Annual Report (18 March 1986 to 31 March 1987*, BBN Laboratories Inc., Cambridge, Ma., Technical Report No. 6636, October 1987.

A. Haas, "Possible Events, Actual Events, and Robots," *Computational Intelligence*, Vol. 1., No. 2, May 1985.

A. Haas, "A Syntactic Theory of Belief and Action," *Artificial Intelligence*, Vol. 28, No. 3, May 1986.

A. Haas, "The Case for Domain-Specific Frame Axioms," *The Frame Problem*, Workshop on Logical Approaches to the Frame Problem, University of Kansas, University of Kansas, 1987.

E. Hinrichs, "A Compostional Semantics for Directional Modifiers in English - Locative Case Reopened," *Proceedings of the 11th International Conference on Computational Linguistics*, Bonn, West Germany, August 1986.

E. Hinrichs and L. Polanyi, "Pointing the Way: A Unified Treatment of Referential Gesture in Interactive Discourse," *Proceedings of the 22nd Annual Meeting of the Chicago Linguistics Society*, Chicago, Il., 1986.

L. Polanyi, "A Theory of Discourse Structure and Discourse Coherence," *Proceedings of the 21st Annual Meeting of the Chicago Linguistics Society*, University of Chicago, Chicago, Il., 1985.

L. Polanyi, *The Linguistic Discourse Model: Towards a Formal Theory of Discourse Structure*, BBN Laboratories Inc., Cambridge, Ma., BBN Technical Report No. 6409, November 1986.

L. Polanyi, *A Formal Syntax of Discourse*, Technical Report of the Center for the Study of Reading, University of Illinois at Urbana-Champaign, 1986.

R. Scha, B. Bruce, and L. Polanyi, "Discourse Understanding," in *Encyclopedia of Artificial Intelligence*, S. C. Shapiro (ed.), John Wiley and Sons, New York, 1986 (also appears as Technical Report No. 391, Center for the Study of Reading, University of Illinois at Urbana-Champaign, 1986).

J. Schmolze, "Physics for Robots," *Proceedings of the Conference of the American Association for Artificial Intelligence*, Philadelphia, Pa., August 1986.

J. Schmolze, *Physics for Robots*, BBN Laboratories Inc., Cambridge, Ma., Technical Report No. 6222, September 1987.

C. Sidner, "Plan Parsing for Intended Response Recognition in Discourse," *Computational Intelligence*, Vol. 1, No. 1, March 1985.

C. Sidner, "Discourse Structure and the Proper Treatment of Interruptions," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, Ca., August 1985.

C. Sidner, "Intentions, Attention and the Structure of Discourse," *Computational Linguistics*, Vol. 12, No. 3, 1986.

C. Sidner and B. Grosz, "Plans in Discourse," *Discourse, Communication and Intention*, MIT Press, Cambridge, Ma., 1988.

N. S. Sridharan, *Semi-Applicative Programming: Examples of Context-Free Recognizers*, BBN Laboratories Inc., Cambridge, Ma., BBN Technical Report No. 6135, January 1986.

M. Vilain, "The Restricted Language Architecture of a Hybrid Representation System," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, Ca., August 1985.

M. Vilain (with H. Kautz), "Constraint Propagation Algorithms for Temporal Reasoning," *Proceedings of the Conference of the American Association for Artificial Intelligence*, Philadelphia, Pa., August 1986.

M. Vilain, *Heterogenous Concurrency in a MIMD Truth Maintenance System*. Position paper accepted for presentation at the AAAI Symposium on Parallel Models of Intelligence, March 1988.

8.3 Forthcoming Papers

A. Haas, "Sentential Semantics for Propositional Attitudes," accepted for publication in *Computational Intelligence*.

C. Sidner and B. Grosz, "Large Distributed Know-How and Acting: Research On Collaborative Planning", to appear in a book on selected papers from the Workshop on Distributed Artificial Intelligence, Lake Arrowhead, Ca., 1988.

B. Grosz, M. Pollack, and C. Sidner, "Computational Models of Discourse," in *Foundations of Cognitive Science*, M. Posner (ed.), 1989.

Official Distribution List

Contract N00014-85-C-0079

	Copies
Scientific Officer Head, Information Sciences Division Office of Naval Research 800 North Quincy Street Arlington, VA 22217-5000 Attn: Dr. Alan Meyrowitz	1
Administrative Contracting Officer Defense Contracts Adm. Services 495 Summer Street Boston, MA 02210-2184 Attn: Mr. Frank Skieber	1
Director, Naval Research Laboratory Attn: Code 2627 Washington, D.C. 20375	1
Defense Technical Information Center Bldg. 5 Cameron Station Alexandria, Va. 22314	12